VŠB – Technická univerzita Ostrava
Fakulta strojní
Katedra automatizační techniky a řízení

Návrh a implementace automatizovaného procesu testování software

Design and Implementation of Automated Software Testing Process

Student: Bc. Lucie Koukolíčková

Vedoucí diplomové práce: Ing. Pavel Smutný, Ph.D.

Ostrava 2019

VŠB - Technická univerzita Ostrava
Fakulta strojní
Katedra automatizační techniky a řízení

Zadání diplomové práce

Student: **Bc. Lucie Koukolíčková**

Studijní program: N2301 Strojní inženýrství

Studijní obor: 3902T004 Automatické řízení a inženýrská informatika

Téma: Návrh a implementace automatizovaného procesu testování software
Design and Implementation of Automated Software Testing Process

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Popište metodiku testování software a zaměřte se na problematiku automatizace aktivit od vývoje testovacích scénářů, jejich implementaci do testovacích a verifikačních skriptů a možnosti využívání nástrojů pro automatizované testování software.
2. Popište dostupné prostředí pro testování zdrojových kódů.
2. Pro jeden příklad užití popište vybrané metody testování a navrhněte kritéria pro posouzení výhod a nevýhod obou postupů (automatizovaného a manuálního procesu).
4. Vytvořte slovníček pojmů a terminologie k dokumentaci pro specifikaci testování.
5. Zhodnoťte dosažené výsledky a navrhněte směry dalšího řešení.

Seznam doporučené odborné literatury:

PATTON, Ron, 2002. *Testování softwaru.* Praha: Computer Press. Programování. ISBN 80-7226-636-5.
ROUDENSKÝ, Petr a Anna HAVLÍČKOVÁ, 2013. *Řízení kvality softwaru: průvodce testováním.* Brno: Computer Press. ISBN 978-80-251-3816-8.
BOROVCOVÁ, Anna, 2008. *Testování webových aplikací.* Praha. Diplomová práce. Univerzita Karlova v Praze.
ROUDENSKÝ, Petr, 2016. *Kvalita softwaru: teorie a praxe.* Prostějov: Computer Media, ISBN 978-80-7402-294-4.
VESELOVSKÝ, Eduard, 2014. *Přehled nástrojů pro automatické testování aplikací.* Plzeň. Bakalářská práce. Západočeská univerzita v Plzni.
BUREŠ, Miroslav, Miroslav RENDA, Michal DOLEŽEL, Peter SVOBODA, Zdeněk GRÖSSL, Martin KOMÁREK, Ondřej MACEK a Radoslav MLYNÁŘ, 2016. *Efektivní testování softwaru: klíčové otázky pro efektivitu testovacího procesu.* Praha: Grada. Profesionál. ISBN 978-80-247-5594-6.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Pavel Smutný, Ph.D.**

Datum zadání: 21.12.2018

Datum odevzdání: 20.05.2019

doc. Ing. Renata Wagnerová, Ph.D.
vedoucí katedry

VYSOKÁ ŠKOLA BÁŇSKÁ TECHNICKÁ UNIVERZITA Fakulta strojní -5- OSTRAVA

prof. Ing. Ivo Hlavatý, Ph.D.
děkan fakulty

Místopřísežné prohlášení studenta

Prohlašuji, že jsem celou diplomovou práci včetně příloh vypracoval samostatně pod vedením vedoucího diplomové práce a uvedl jsem všechny použité podklady a literaturu.

V Ostravě dne 20. května 2019

Podpis studenta

Prohlašuji, že:

- jsem si vědom, že na tuto moji závěrečnou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. Zákon o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (dále jen Autorský zákon), zejména § 35 (Užití díla v rámci občanských či náboženských obřadů nebo v rámci úředních akcí pořádaných orgány veřejné správy, v rámci školních představení a užití díla školního) a § 60 (Školní dílo),
- beru na vědomí, že Vysoká škola báňská – Technická univerzita Ostrava (dále jen „VŠB-TUO“) má právo užít tuto závěrečnou diplomovou práci nekomerčně ke své vnitřní potřebě (§ 35 odst. 3 Autorského zákona),
- bude-li požadováno, jeden výtisk této diplomové práce bude uložen u vedoucího práce,
- s VŠB-TUO, v případě zájmu z její strany, uzavřu licenční smlouvu s oprávněním užít dílo v rozsahu § 12 odst. 4 Autorského zákona,
- užít toto své dílo, nebo poskytnout licenci k jejímu využití, mohu jen se souhlasem VŠB-TUO, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly VŠB-TUO na vytvoření díla vynaloženy (až do jejich skutečné výše),
- beru na vědomí, že - podle zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů - že tato diplomová práce bude před obhajobou zveřejněna na pracovišti vedoucího práce, a v elektronické podobě uložena a po obhajobě zveřejněna v Ústřední knihovně VŠB-TUO, a to bez ohledu na výsledek její obhajoby.

V Ostravě dne 20. května 2019

Koukolíčková

.....

Podpis autora práce

Jméno a příjmení autora práce: Lucie Koukolíčková

Adresa trvalého pobytu autora práce: Jižní 29, 695 01 Hodonín

ANOTACE DIPLOMOVÉ PRÁCE

KOUKOLÍČKOVÁ, L. *Návrh a implementace automatizovaného procesu testování software: diplomová práce*. Ostrava: VŠB - Technická univerzita Ostrava, Fakulta strojní, Katedra automatizační techniky a řízení, 2019, 60 s. Vedoucí práce: Smutný, P.

Diplomová práce se zabývá teorií testování software a rozebírá základní charakteristiky testování. V rámci popisu metodiky testování software se zaměřuje na problematiku automatizace aktivit od vývoje testovacích scénářů, jejich implementaci do testovacích a verifikačních skriptů až po možnosti využívání nástrojů pro automatizované testování software.

V rámci praktického příkladu byl zpracován slovníček pojmů a terminologie z této oblasti a dále jsou demonstrovány jednotlivé typy testů, možnosti automatizace a výhodnost/nevýhodnost jejího zavedení.

Klíčová slova: testování software, zajištění kvality, údržbové testování, prostředí automatizovaného testování, metodika testování, RUP, scrum, agilní testování, automatizované testování software, generování a výběr automatizovaných testovacích případů, generování a výběr dat pro automatizované testy, chyba software, procesní modely vývoje software

ANNOTATION OF MASTER THESIS

KOUKOLÍČKOVÁ, Lucie. *Design and Implementation of Automated Software Testing Process: Master Thesis*. Ostrava: VŠB-Technical University of Ostrava, Faculty of Mechanical Engineering, Department of Control Systems and Instrumentation, 2019, 60 p. Thesis head: Smutný, P.

Master thesis is focused on theory of software testing and explains the basics characteristics of testing. In the explanation of testing methods the main point is focused on automated testing. It starts with the design of testing scenarios, their implementation into the testing and verification scripts and at the end of process the possibilities of using automated testing tools.

In the practical example the vocabulary of specific terms from testing process was created and there are shown various types of tests, automation possibilities and advantages/disadvantages of using automation in the process of testing.

Key words: software testing, quality assurance, maintenance testing, automated test environment, testing methodology, RUP, scrum, agile testing, automated software testing, automated test case generation and selection, automated test data generation and selection, software bug, software development process models

Obsah

Seznam použitých zkratek a symbolů 8
Úvod 9
1 Testování software 10
1.1 Testování - základní charakteristika 10
1.2 Chyba - základní charakteristika 12
1.3 Aktivity vedoucí k plánování testování 15
2 Metodika testování software 18
2.1 Testování v tradičním vývoji 18
2.2 Testování v agilním vývoji 20
2.3 Testování ad hoc a exploratorní testování 21
3 Metody a principy testování 23
3.1 Úrovně testování 23
3.2 Statické testy 24
3.2.1 Testování dokumentace 24
3.2.2 Testování zdrojového kódu 25
3.3 Dynamické testy 27
3.4 Testování metodou Smoke test 27
3.5 Testování metodou Black box 27
3.6 Testování metodou White box 28
3.7 Testování metodou Grey box 29
3.8 Testování metodou End-to-end 30
3.9 Slovník 30
4 Automatizované testování software 32
5 Skutečná podoba testování 36
5.1 Příklad užití 37
5.2 Black box test 42
5.3 Grey box test 44
5.4 White box test 47
5.5 Automatizace testů 48
5.6 Zhodnocení 51
6 Závěr 52
7 Seznam použité literatury 55
8 Příloha 1 - Slovník 57

Seznam obrázků

Obrázek 1 - Definice pojmů *přesný* a *správný* (Patton, 2002) 11
Obrázek 2 - Efektivní hladina testování (Patton, 2002) 12
Obrázek 3 - Příčiny chyb, zdroj z roku 2001 (Patton, 2002) 14
Obrázek 4 - Příčiny chyb, zdroj z roku 2010 (Roudenský, 2013) 14
Obrázek 5 - Náklady na opravu chyby (Patton, 2002) 15
Obrázek 6 - Trojúhelník kvality (Roudenský, 2013) 15
Obrázek 7 - Hierarchický rozpad činností (Roudenský, 2013) 17
Obrázek 8 - Schematické vyjádření procesu RUP (Vondrák, 2002) 19
Obrázek 9 - Testování metodou Black box (Steiner, 2005) 27
Obrázek 10 - Testování metodou White box (Steiner, 2005) 29
Obrázek 11 - Testování metodou Grey box (Ramasubbarayalu, 2017) 30
Obrázek 12 - Náhled webu 31
Obrázek 13 - Návratnost investice do automatizace v čase (Bureš, 2016) 34
Obrázek 14 - Panel vložení nabídky 37
Obrázek 15 - Nastavení limitů pro uživatele 38
Obrázek 16 - Diagram průběhu zavedení nabídky 39
Obrázek 17 - Testovací scénář Black box 43
Obrázek 18 - Testovací scénář Grey box (1. část) 45
Obrázek 19 - Testovací scénář Grey box (2. část) 46
Obrázek 20 - Ukázka přípravy automatického testu v software Jubula 50
Obrázek 21 – Ukázka scénáře prováděného automatem 50

Seznam tabulek

Tabulka 1 - Cíle testování (Bureš, 2016) 17
Tabulka 2 - Rozpad testovacích případů na scénáře 40
Tabulka 3 - Soubor unit testů pro panel Zadání nabídky 47
Tabulka 4 - Příklad hodnot pro automatické zakládání nabídek 49
Tabulka 5 - Zhodnocení manuálního a automatizovaného Black box testu 51

Seznam použitých zkratek a symbolů

CSV	Comma-separated values (hodnoty oddělené čárkami; jednoduchý formát souboru určený pro výměnu tabulkových dat)
IBM	International Business Machines Inc.
IEC	International Electrotechnical Commission (Mezinárodní elektrotechnická komise)
ISO	International Organisation for Standartization (Mezinárodní organizace pro normalizaci)
MIT	Massachusetts Institute of Technology
RUP	Rational Unified Process
UAT	User Acceptance Testing
XLS	přípona souborů vytvořených v programu Microsoft Excel

Úvod

V českých podmínkách je povědomí o testování software teprve v rozbíhajícím se procesu. Kvalitní odborné literatury pro začátečníky v českém jazyce je pomálu, studijní či rekvalifikační obor tohoto druhu neexistuje. A i když se objevují snahy o to, aby byla vybudována jakási testerská komunita, která bude sdílet poznatky, rozšiřovat obzory, pořádat přednášky či konference, vše je zatím jen v minimálním měřítku. Zvláště pokud provedeme srovnání s komunitou, která se pohybuje okolo programování, vývoje a distribuce aplikací.

Ačkoliv následující stránky nepředstavují plný a podrobný vhled do problematiky, nabízejí základní teoretické a posléze i praktické informace ze světa testerů a testování software. První část je zaměřena především na obecné poznatky o testování, jeho podstatu, cíle a strukturu přípravy. Druhá část rozebírá různé možnosti přístupu k testování - tzv. metodiky - jejich princip, hlavní myšlenky a vhodnost uplatnění podle typu projektů. Následující kapitola již podrobněji rozebírá jednotlivé typy testů, jejich vlastnosti, vhodnost či nevhodnost pro ověření konkrétních situací. V návaznosti na dosud získané informace je pozornost zaměřena také na automatizované testování. Tato kapitola se zabývá nejmladší a momentálně nejatraktivnější částí testování software. V oblasti automatického testování zatím nejsou striktně stanoveny hranice ani pokryty všechny možnosti využití, což skýtá velký prostor novým experimentům.

Po teoretické přípravě následuje na praktickém příkladu provedená ukázka testerské práce. Abychom mohli vykonat zhodnocení, jsou stanovena kritéria, která budeme primárně sledovat jak u manuálních, tak u automatizovaných testů. Po té jsou představeny nástroje, které budou v této kapitole využívány. Následně se text zaměří na samotnou praktickou demonstraci.

Nejprve je prostor věnován manuálnímu testování. Posléze je provedena transformace manuálního procesu do automatického a jsou zhodnoceny přínosy/nevýhody tohoto převodu.

Pro větší názornost jsou použity vizuální podklady reálných vstupů a výstupů ve zvolených nástrojích. Jedná se o platformu Jira s nástavbou Zephyr (pro evidenci prací a úkolů testovacího týmu s vazbou na vývojový a analytický tým), software Jubula (nástroj pro automatické testování grafického rozhraní) a jako prostředník je použit program Jenkins (spouští automatické testování připravené v software Jubula a jeho výsledky zapisuje do platformy Jira).

Troufalým cílem tohoto textu tak je nejen představit obecné teoretické poznatky s praktickou ukázkou, ale být jakýmsi vstupním průvodcem do světa testování. Z tohoto důvodu bylo také během tvorby textu nashromážděno základní pojmosloví užívané v testerské praxi a v komunikaci mezi testovacím a vývojovým týmem. Slovník je zařazen jako příloha na konci práce. Pokud byly některé z pojmů použity v průběhu textu, je jejich význam vysvětlen okamžitě v místě užití.

Druhým očekávaným výsledkem je zhodnocení užitečnosti, popřípadě vhodnosti implementace automatického testování do procesu vývoje software. I přesto, že jeho aplikace bude předvedena pouze na jednom případu užití, spolu s teoretickými podklady je možné získané výsledky zobecnit a porovnat vůči testování manuálnímu.

1 Testování software

Definic testování nalezneme v literatuře mnoho. Záleží na době vzniku, na testovaném odvětví, a také na pohledu autora na dané téma.

V osmdesátých letech 20. století bylo testování vnímáno jako procházení aplikace za účelem nalezení chyb. V devadesátých letech se definice posunula k procesu práce s produktem nebo jeho částí za specifických podmínek. Výsledky se zaznamenávaly a vyhodnocovaly. Na přelomu století již bylo testování chápáno jako prověřování funkčnosti produktu za účelem přezkoumání kvality s ohledem na požadavky zadavatele. (Uspenskiy, 2010)

Pro účely této práce budeme vycházet z definice popisující testování jako „*proces řízeného spouštění softwarového produktu s cílem zjistit, zda splňuje specifikované či implicitní potřeby uživatelů"* (Roudenský, 2013).

Můžeme tedy obecně říci, že testování je řízený proces, během kterého zkoumáme vlastnosti a kvalitu provedení výsledného produktu, a hodnotíme jeho schopnost naplnit očekávanou funkčnost.

Je třeba si také uvědomit, že ačkoliv je testování aktivita spadající pod řízení kvality, tato aktivita sama o sobě kvalitu výsledného produktu nezvyšuje, ani ji nezajišťuje. Jedná se pouze o nástroj poskytující pro tuto činnost vhodné vstupy. (Roudenský, 2013)

Pro posuzování kvality software jsou, stejně jako například ve strojírenské praxi, stanoveny normy. Do roku 2016 byla v platnosti norma ISO/IEC 9126-1, která stanovovala šest základních charakteristik kvality pro systémy obsahující software. Ačkoliv byla tato norma bez náhrady zrušena (TECHNOR print, 2018), můžeme si na obsažených charakteristikách definovat očekávání, jež by měl splňovat dobře naprogramovaný a vhodně otestovaný software (Vaníček, 2004):

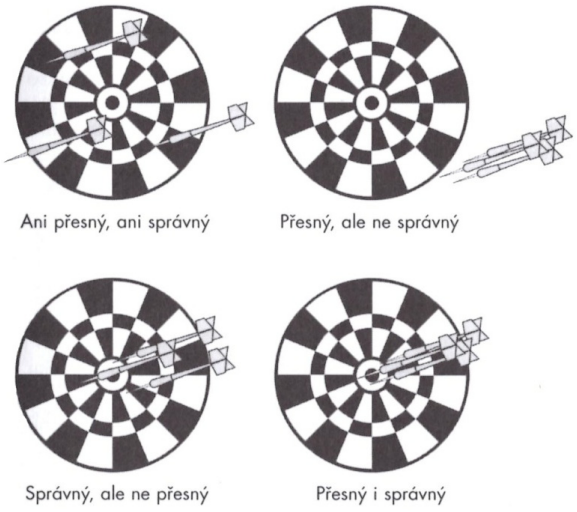
- *funkčnost* - schopnost produktu zabezpečit požadované funkce,
- *bezporuchovost* - schopnost produktu zajistit za daných podmínek požadovanou úroveň výkonu a poskytovaných služeb,
- *použitelnost* - schopnost produktu být využíván při přiměřené míře úsilí potřebného na seznámení se s jeho možnostmi a jeho běžné provozování v daných podmínkách,
- *účinnost* - schopnost produktu zajistit služby s přiměřenými nároky na zdroje systému a v přiměřené době,
- *udržovatelnost* - schopnost produktu být v průběhu používání měněn s cílem přizpůsobení požadavkům uživatele, odstranění zjištěných nedostatků, rozvoje a zlepšování funkcí nebo změny prostředí, v kterém má pracovat,
- *přenositelnost* - schopnost produktu pracovat na datové i procesní úrovni s jinými systémy včetně těch, které pracují na jiných platformách.

1.1 Testování - základní charakteristika

Dříve než se začneme hlouběji zabývat samotnou podstatou testování, je třeba definovat pojmy, které tvoří základnu pro správné pochopení důležitosti testování. Jelikož se často jedná o pojmy významově blízké, může u nich docházet ke špatné

interpretaci. Definici tedy provedeme ve dvojicích, které pomohou lépe pochopit skutečný význam i vzájemné rozdíly jednotlivých slov (Patton, 2002):

- *Přesnost a správnost* - pro pochopení těchto termínů použijeme názornou demonstraci hrou šipky (viz Obrázek 1). Jejím cílem je zasáhnout střed terče a získat nejvyšší počet bodů. Obrázek vlevo nahoře zobrazuje zásah, který není ani přesný, ani správný. Vpravo nahoře vidíme zásah, který je přesný (šipky jsou blízko sebe), ale není správný (mimo cíl). Levý dolní obrázek představuje správný hod (na střed terče), ale chybí přesnost. Vpravo dole vidíme ideální situaci, kdy je zásah přesný a správný.



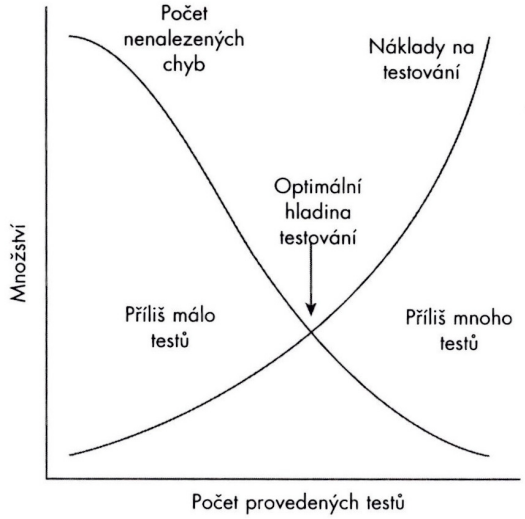
Obrázek 1 - Definice pojmů *přesný* a *správný* (Patton, 2002)

- *Verifikace a validace* - tato dvě slova jsou si zdánlivě velmi podobná, ale jedná se o dva zcela odlišné významy. Verifikace je proces, který má za cíl potvrdit, že předmět zkoumání vyhovuje zadané specifikaci. Validace na druhou stranu zhodnocuje, zda předmět zkoumání splňuje požadavky uživatele.
- *Kvalita a spolehlivost* - pojmy, jež bývají mnohdy významově zaměňovány v domnění, že jde o totéž. I zde je však podstatný rozdíl. Kvalita vyjadřuje, jak dalece výsledný produkt vyhovuje potřebám zákazníka, zatímco spolehlivost indikuje, jak často produkt havaruje nebo jak často poškodí data v něm obsažená.
- *Testování a zajišťování kvality* - pojmy představují dva procesy, které se v jistém rozsahu vzájemně překrývají. Testování má za úkol nacházet chyby v produktu a zajišťovat jejich nápravu. Proces zajišťování kvality má za úkol vytvářet a prosazovat vhodné standardy a metody, které poslouží ke zdokonalení procesu vývoje a bude se tak předcházet vzniku chyb.

Výše zmíněné pojmy a jejich stoprocentní naplnění představují ideální stav. V praxi však není vždy možné splnit cíle v plném rozsahu. Postupem času se tak vyvinula určitá pravidla, nebo můžeme říci obecné pravdy, které v různém rozsahu provází každý nově vznikající projekt a charakterizují jeho možné nedostatky. Ve zjednodušené podobě je můžeme shrnout do následujícího seznamu (Patton, 2002):

- žádný program není možné otestovat kompletně,
- testování software je postavené na riziku,
- testování nikdy nemůže prokázat, že chyby neexistují,
- čím více chyb najdeme, tím víc chyb v software je,
- paradox pesticidů (čím více daný software testujeme, tím více se stává vůči testování imunní),
- ne všechny nalezené chyby se opraví,
- je těžko říci, kdy je chyba chybou,
- specifikace produktu nejsou nikdy konečné,
- testeři software nejsou těmi nejoblíbenějšími členy projektového týmu,
- testování software je přesná technická disciplína.

Shrneme-li uvedené informace, je zřejmé, že testování představuje kompromis. Je důležité najít efektivní hladinu testování, která zajistí odevzdání kvalitního produktu v určeném čase a s přiměřenými náklady. Kde se tato hladina nachází, obecně znázorňuje níže uvedený graf (viz Obrázek 2).



Obrázek 2 - Efektivní hladina testování (Patton, 2002)

1.2 Chyba - základní charakteristika

V oficiálních zdrojích se můžeme setkat s vlastností výpočetních systémů zvanou dependabilita. Jedná se o schopnost výpočetního systému dodávat službu, které lze oprávněně důvěřovat. Koncept dependability tvoří tři části - prostředky k jejímu dosažení, atributy a hrozby. Jako hrozby dependability označujeme (Roudenský, 2013):

- *defekty* (jedná se o následek pochybení člověka, jsou příčinami chyb),
- *chyby* (jsou stavy systému a mohou vést k selhání),
- *selhání* (nesoulad mezi aktuálním a specifikovaným chováním systému).

V praxi je však mnohem častěji používáno neformální a zcela obecné označení „chyba“ (nebo originální anglický pojem „bug“).

Abychom mohli přesněji definovat pojem chyba, je třeba nejprve popsat, ve vztahu k čemu je chyba shledána. Při vývoji software je obvykle základním informačním prvkem specifikace produktu. Její text popisuje, co je od software očekáváno, jak má vypadat a co naopak dělat nemá a nesmí. Vůči tomuto dokumentu je pak možno chybu popsat těmito pravidly (Patton, 2002):

- Software nedělá něco, co by podle specifikace produktu dělat měl.
- Software dělá něco, co by podle údajů specifikace produktu dělat neměl.
- Software dělá něco, o čem se produktová specifikace nezmiňuje.
- Software dělá něco, o čem se produktová specifikace nezmiňuje, ale měla by se zmiňovat.
- Software je obtížně srozumitelný, těžko se s ním pracuje, je pomalý, nebo (podle názoru testera software) jej koncový uživatel nebude považovat za správný.

Termín chyba bývá nahrazován mnoha jinými slovy, podle skupiny, ve které se zrovna nacházíme. Z používaných výrazů si uveďme pro příklad (Patton, 2002):

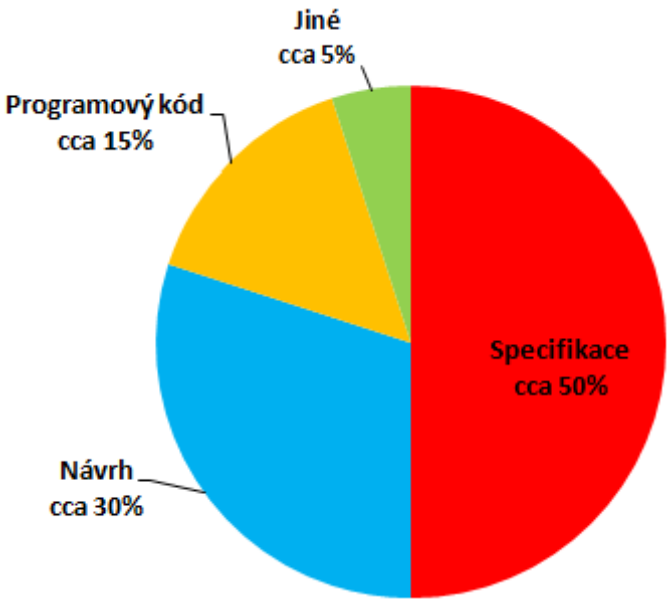
- Vada	- Odchylka
- Závada	- Selhání
- Problém	- Nekonzistence
- Omyl	- Vlastnost
- Událost	- Chyba
- Anomálie	

V prvním plánu významu jsou tato slova zaměnitelná. Vždy se jedná o označení problému, který se v testovaném produktu vyskytuje. V druhém plánu je však možné vycítit jisté odlišnosti, dle kterých můžeme používaná označení chyby rozdělit do několika skupin (Patton, 2002):

- *závada, selhání, vada*: jsou vnímány jako označení situace, která je skutečně závažná, či dokonce nebezpečná,
- *anomálie, událost, odchylka*: tato označení nevyvolávají tak záporný dojem, evokují spíše nedomyšlenou činnost nebo neúmyslnou chybu,
- *problém, omyl, chyba*: tyto pojmy jsou nejobecněji používané.

Ačkoliv je český jazyk bohatý a umožňuje mnohé variace pro označení vzniklého problému, je zcela evidentní, že slovní označení je pouze volbou mluvčího. Ve výsledku se totiž jedná o vždy stejné sdělení - nastal problém a je třeba ho vyřešit.

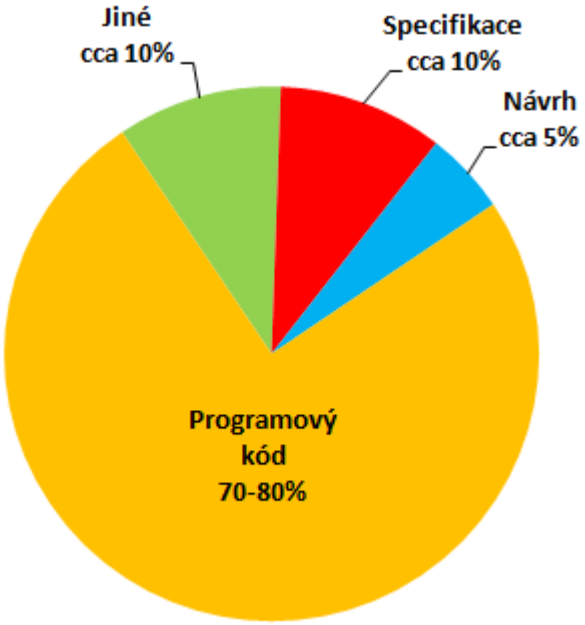
Na základě studií provedených na řadě projektů různých rozsahů do roku 2001, se prokázalo, že největším viníkem vzniku chyb byla specifikace (viz Obrázek 3).



Obrázek 3 - Příčiny chyb, zdroj z roku 2001 (Patton, 2002)

Tato skutečnost měla několik důvodů - v mnoha projektech specifikace vůbec neexistovala, v jiných sice existovala, ale neustále se měnila nebo nebyla dostatečně konzultována.

Budeme-li vycházet ze zdrojů z roku 2010, zjistíme, že příčina chyby se markantně (70 - 80%) přesunula do oblasti vzniku zdrojového kódu (viz Obrázek 4). Tuto změnu je možné přisuzovat užívání vhodně zvolených metodik, dodržování norem a také stále se rozvíjejícím nástrojům pro analýzu a návrh systému (Roudenský, 2013).

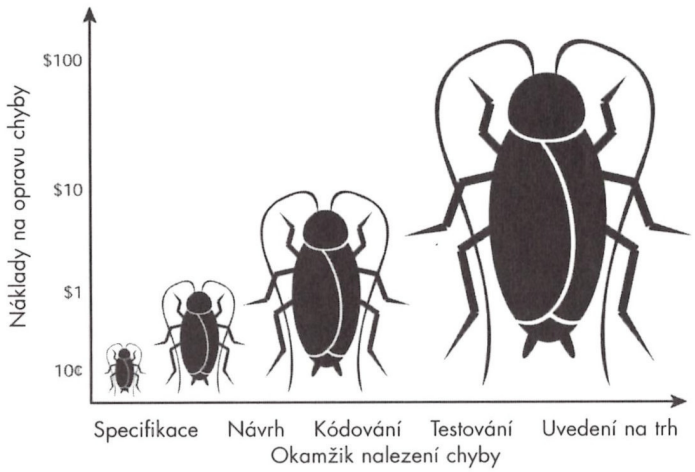


Obrázek 4 - Příčiny chyb, zdroj z roku 2010 (Roudenský, 2013)

Z manažerského hlediska je třeba také uvažovat o nákladnosti jednotlivých chyb, respektive ceně jejich opravy ve vztahu k období, kdy je chyba nalezena. Na uvedeném

obrázku (viz Obrázek 5) je symbolicky znázorněno, že se nákladnost zvyšuje až k desetinásobku.

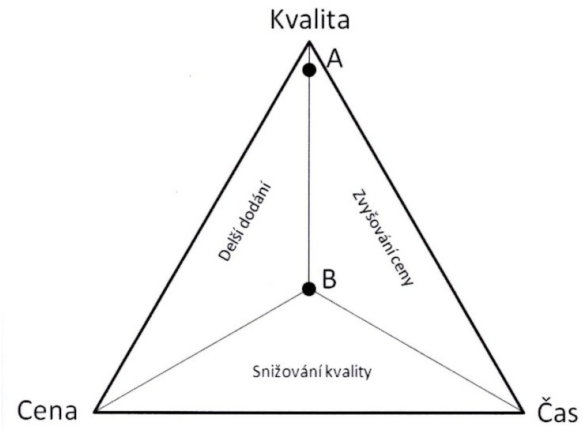
Nalezneme-li nepřesné definice a potenciální problémy ve funkční specifikaci, je oprava zanedbatelná časově a tedy i finančně. V následných etapách se cena zvyšuje úměrně k „počtu rukou“, kterými musí problém projít, než je vyřešen. Nastane-li situace, že problém odhalí až koncový uživatel, je jeho oprava pochopitelně nejnákladnější, nemluvě o jisté diskreditaci tvůrčího (dodavatelského) týmu.



Obrázek 5 - Náklady na opravu chyby (Patton, 2002)

1.3 Aktivity vedoucí k plánování testování

Plánování testování je zahrnuto již v základní přípravě projektu. Podle náročnosti chystaného programu (změny), finančního zajištění projektu, počtu přidělených pracovníků a také dostupných nástrojů, je třeba v harmonogramu prací vytvořit dostatečně velký časový slot, ve kterém bude možné provést relevantní testování. Také je nutné dohodnout, co přesně a v jakém rozsahu se má testovat, aby bylo dosaženo požadovaného výsledku. V zobecněné podobě můžeme tento základní problém znázornit tzv. trojúhelníkem kvality (viz Obrázek 6). Bod A zde představuje drahý, velmi kvalitní systém s dlouhou dobou dodání, bod B naopak každému atributu věnuje stejné úsilí, ale ve výsledky dochází ke zbytečným ztrátám na všech stranách (Roudenský, 2013). Je tedy třeba nalézt takový bod, který dostatečně vyhovuje všem stranám, přičemž vždy se mění jeden parametr na úkor ostatních.



Obrázek 6 - Trojúhelník kvality (Roudenský, 2013)

Pokud je vývoj veden takříkajíc z čistého listu, tedy vzniká něco zcela nového, je pochopitelně nutné věnovat přípravě scénářů a samotnému testování mnohem více času a péče než v případě změny v již existujícím projektu. V takové situaci je obvykle již část práce hotová z předchozího vývoje a je třeba jen vypíchnout zasažené oblasti, zrevidovat stávající testovací scénáře do nové podoby zahrnující provedenou změnu a připravit případné nové scénáře pro nově implementované prvky. V každém případě je však nutné připravit tzv. plán testování, z kterého vycházejí všechny následné kroky. Obecně je hlavním obsahem tohoto dokumentu (Roudenský, 2013):

- *cíl testování* - záměr a cíl testování (prokázání funkčnosti, soulad s požadavky, odhalení defektu,…),
- *rámec testování* - co má/nemá být předmětem testování,
- *způsob testování, strategie* - jak bude testování prováděno (metody, nástroje, přístup, techniky,…),
- *rizika testovaného systému* - identifikace oblastí, u nichž je vyšší pravděpodobnost výskytu defektů,
- *potřebné zdroje* - lidé, hardware, požadavky na prostředí,
- *artefakty testování* - testovací případy a testovací data, skripty (v případě automatizace), zaznamenané defekty, souhrnná hlášení o provedených testech,
- *plánované aktivity a úlohy* - detailní seznam všech úloh, z nichž se proces testování skládá, jež jsou dále rozepsány na jednotlivé dílčí úlohy,
- *harmonogram testování* - od analýzy, návrh testovacích případů po vyhodnocení výsledků provedených testů,
- *metriky* - prostředky měření a sledování postupu a úspěch celého projektu i vlastního testování,
- *identifikace výstupních kritérií* - definice podmínek, jejichž splnění je nutným předpokladem pro dokončení určité aktivity testování či testovacího plánu jako celku,
- *identifikace kritérií pro pozastavení a obnovení testování* - za jakých podmínek je nutné přerušit provádění testování (typicky vysoký počet nalezených defektů, kritický defekt zabraňující testování, problém s testovacím prostředím) a kdy je možné v pozastaveném testování pokračovat,
- *rizika ohrožující testování* - rozpočet, zdroje a jejich dostupnost, změny časového rozvrhu,…

Ve výše zmíněném plánu testování se nacházejí dva body, které je třeba hlouběji rozebrat, neboť svým rozsahem zaštiťují body ostatní. Nejprve se zaměříme na podrobnější specifikaci cílů testování.

Pokud bychom se rozhodli postavit cíl testování na základní myšlence „ať to funguje“, dostaneme se v průběhu času do úzkých. Jelikož je tento cíl velmi obecný, nemáme možnost přesně určit, co očekáváme od testů a postrádáme základní informaci o tom, co je skutečně důležité. Navíc pomocí testů nejsme schopni zajistit funkčnost systému, můžeme pouze odhalit defekty (tedy zjednodušeně řečeno „že to nefunguje“). Také by tato formulace mohla vzbudit dojem, že testovací tým je zodpovědný za kompletní

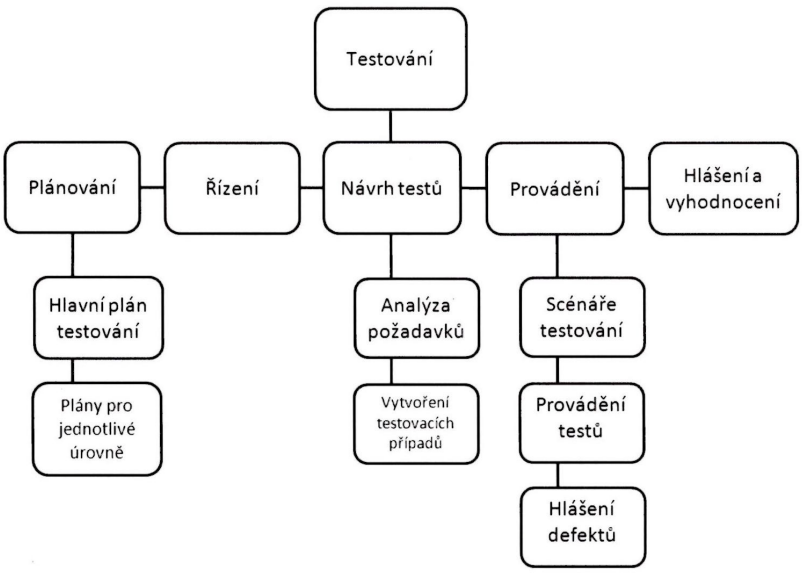
otestování bez ohledu na zdroje, které reálně dostane k dispozici, a na splnění vstupních předpokladů, které k tomu potřebuje. (Bureš, 2016)

Je tedy zřejmé, že při tvorbě cílů testování musíme rozebrat očekávané výsledky hlouběji na základě priorit zákazníka/uživatele systému, klíčových změn a s ohledem na rizika testovaného systému. Výsledné shrnutí cílů testování pak může představovat například tabulka (viz Tabulka 1).

Tabulka 1 - Cíle testování (Bureš, 2016)

ID	Cíle testování
1	**V kritických funkcích systémů nesmí být výpadky**
1.1	Funkce pořizování letenek musí spolehlivě fungovat
1.2	Systém musí být dostupný v režimu 24x7
1.3	...
2	**Datová migrace musí proběhnout v pořádku**
2.1	V systému musí být možné rychle dohledat historii transakcí ze systému, který se používal předtím
2.2	...
3	**Systém musí splňovat normy dané příslušnými úřady**
3.1	Systém musí splňovat normu X
3.2	Systém musí být možné napojit na centrální evidenci letových časů a linek
3.3	...

Máme-li specifikováno, co přesně je třeba testovat, zaměřme se nyní na druhý důležitý bod a tím je hierarchický rozpad činností a určení rolí (viz Obrázek 7). Jedná se o detailní seznam všech úloh, z nichž je složen testovací proces, a které jsou dále rozepsány na jednotlivé dílčí úlohy. Tento rozpis je třeba provést velmi pečlivě, neboť to, co je v rozpise opomenuto, obvykle bude opomenuto i při provádění.



Obrázek 7 - Hierarchický rozpad činností (Roudenský, 2013)

Zároveň s vytvářením seznamu úloh je vhodné rovnou provádět přiřazení na jednotlivé členy týmu. Každý by měl mít jasně definovánu svou roli a nemělo by docházet k překrývání působností. Tak budou jasně vymezeny zodpovědnosti a určeny lidské zdroje.

2 Metodika testování software

Obecně vzato je metodika postup nebo soubor postupů, jak se zachovat v určité situaci. Může zahrnovat nejen přesný sled pokynů, jejichž následováním dojdeme k žádanému cíli, ale také doporučení či návrhy, jak se v té či oné situaci zachovat v případě neočekávaných okolností, co můžeme na základním postupu vylepšit nebo rozvinout, případně kterým jiným směrem se můžeme dostat do stejného následujícího bodu.

Tento popis je velmi široký a pochopitelně v každém odvětví dochází ke konkrétní aplikaci a specifikaci pro danou problematiku. Prvotní metodiky související s naším tématem se začaly objevovat při vzniku softwarového inženýrství, tedy oboru, který se snaží o systematizaci vývoje a údržby software. Můžeme říci, že vznik jak oboru, tak i metodik je důsledkem neustále rychleji probíhajícího rozvoje, který klade vyšší a vyšší nároky na vyvíjené aplikace. Pro udržitelnost nastavených standardů je tedy nezbytně nutné vnášet do pracovních procesů řád a stavět takříkajíc „od základů“, nikoliv zprostřed nebo z několika různých stran s nadějí, že se snad někde všechny části potkají. (Borovcová, 2008)

Všechny tyto skutečnosti samozřejmě platí nejen pro samotný vývoj software, ale také pro jeho testování, které v současné době stále více významově roste a dostává se do popředí zájmu.

2.1 Testování v tradičním vývoji

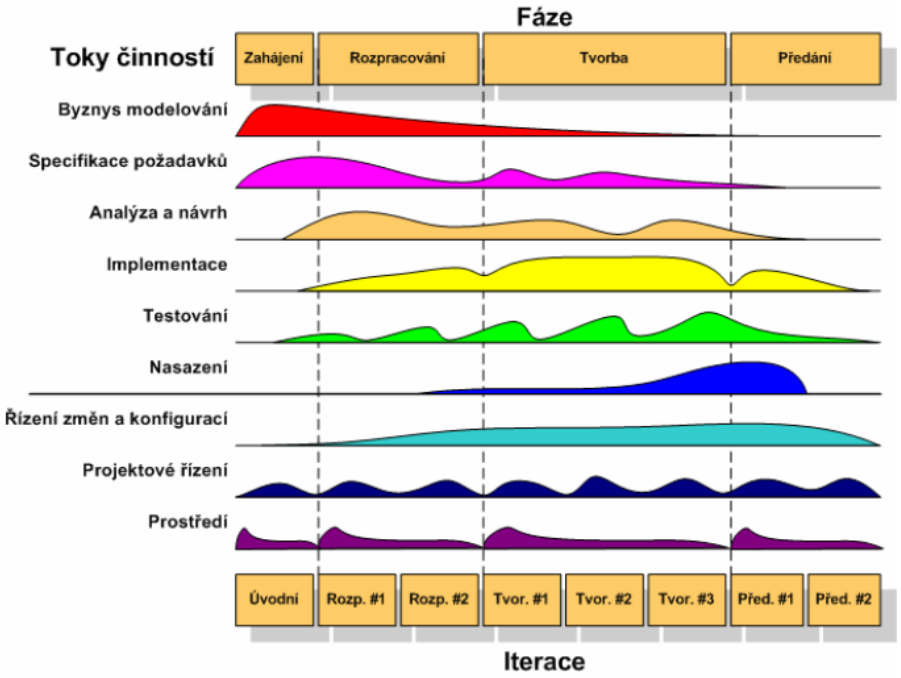
Tradiční testování, jak již název napovídá, je založeno na původních metodikách, které jsou striktně vázány na organizovanost a disciplínu v prováděcím procesu. S rozvojem možností softwarového programování se metodiky doplňovaly, rozvíjely a upravovaly pro jednotlivé situace či procesy. Přesto si stále zachovávají originální myšlenku. Tedy převzít nastolený řád, dodržet předepsané kroky a dojít k stanovenému cíli.

Jedním z typických zástupců tohoto typu testování je Rational Unified Process (RUP). Autorem této metodiky je společnost Rational Software Corporation, která nyní tvoří samostatnou divizi společnosti IBM. Principiálně je metodika určena spíše pro rozsáhlejší projekty, ale teoreticky je možné ji přizpůsobit i pro menší úkoly. Je však samozřejmě třeba zvážit vhodnost a výhodnost aplikace na daný záměr. Myšlenka metodiky je postavena na tzv. šesti nejlepších praktikách (viz Obrázek 8)(Julínek, 2008):

- *iterativní vývoj,*
- *aktivní správa požadavků,*
- *komponentová architektura,*
- *vizuální modelování,*
- *ověřování kvality software,*
- *řízení změn.*

RUP rozděluje vývoj projektu do 4 základních fází, které je dále možno štěpit do více částí nazývaných iterace (viz Obrázek 8)(Julínek, 2008):

- *zahájení* (účel, rozsah projektu),
- *příprava* (analýza požadavků zákazníka, definice základů architektury),
- *konstrukce* (tvorba zdrojových kódů),
- *předávání* (předání zákazníkovi).



Obrázek 8 - Schematické vyjádření procesu RUP (Vondrák, 2002)

Podmínkou započetí nové iterace/fáze je vždy splnění všech definovaných kritérií předchozí iterace/fáze.

Pro testovací oblast, na které je tato metodika víceméně postavena, je zde definováno hned několik rolí a aktivit, které jsou zaměřeny na testování. Projekt prochází mnoha fázemi, testy tak lze neustále rozvíjet a specifikovat. V základu metodika určuje čtyři testerské role a jejich hierarchii - Test manager, Test analytik, Test designer, Tester.

Nejprve je určeno, co a proč se bude testovat a jaká jsou výstupní kritéria testů. Tento úkol vykonává Test manager. Po té je stanoven způsob testování, jaké budou použity nástroje a k jaké konfiguraci aplikace se to bude vztahovat. Tato část spadá do kompetence Test analytika. Test designer podle obdržených informací vypracuje testy, připraví testovací data a vše seskupí do logických celků. Po té, co je dodán produkt, provede Test designer ve spolupráci s testerem prvotní ověření stability produktu metodou Smoke testů. Je li dodaný produkt vhodný k testování započne provádění testů. Tester nastaví požadovanou konfiguraci aplikace a zpracovává jednotlivé testovací scénáře. V případě, že se objeví jakékoliv nesrovnalosti, jsou ihned hlášeny. Závěrečná fáze představuje vyhodnocení testů. Souhrnné hlášení se zaznamenanými problémy je vráceno vývojovému týmu ke zpracování. Tento úkol opět zpracovává Test manager. (Anon., 2018a)

2.2 Testování v agilním vývoji

Již v devadesátých letech se začaly objevovat požadavky na nalezení metodiky vhodné pro malé vývojové týmy, která by byla rychlejší a pružnější než dosavadní klasické způsoby. V roce 2001 byl týmem odborníků sepsán agilní manifest (Highsmith, 2001), který je založen především na nutnosti komunikace a flexibility během celého vývoje.

Základní myšlenky agilního vývoje software shrnují body (Roudenský, 2013):

- *jednotlivci a interakce* před procesy a nástroji,
- *fungující software* před vyčerpávající dokumentací,
- *spolupráce se zákazníkem* před vyjednáváním o smlouvě,
- *reagování na změny* před dodržováním plánu.

Agilní metodiky jsou tedy vhodné především pro menší projekty, kde dochází k častým změnám požadavků a je třeba práci odvádět rychle a efektivně. Je zde doporučováno vytvářet co nejmenší množství dokumentace a věnovat velkou pozornost čitelnosti a srozumitelnosti kódu. Je-li pak třeba provést v kódu opravu, je zásah minimální a následuje pouze drobná úprava v přidružených dokumentech. Z praktického hlediska tak dochází také ke snižování celkových nákladů na vývoj, neboť potřebné zásahy jsou méně pracné a tedy i finančně méně náročné. Jednotlivé fáze agilního vývoje můžeme seřadit do šesti základních kroků (Knesl, 2009a):

1. *nultá iterace* (prvotní analýza, naprogramování základní činnosti, není důležité, co vznikne, podstatné je mít něco, co je možné prezentovat klientovi),
2. *analýza změny* (co se bude implementovat, analýza, design změn),
3. *implementace* požadované vlastnosti,
4. *předvedení* klientovi,
5. *produkt není hotov*, zpět na 2. bod,
6. *produkt je hotov*, následuje údržba, rozvoj.

Body 2-4 jsou chápány jako jedna iterace. Opakují se tak dlouho, dokud není vývoj ukončen.

Za hlavní přínos agilních metodik testování můžeme považovat důraz na užívání testovacích nástrojů. Automatizované testování nejen, že vnáší do vývoje zrychlení a větší preciznost při opakovaných testech, ale část přípravy automatických testů mohou rovnou zpracovat programátoři, protože přípravy testovacích scénářů jsou daleko podobnější programování než testování. Díky tomuto přesunu úkolů pak mají testeři více času na provádění úkolů, které nelze nahradit automatizovanými testy a jejich pozornost není rozptylována rutinním opakováním nutných ověření.

Často bývají myšlenky agilního vývoje software špatně pochopeny a prezentovány jako metoda, která vůbec nedokumentuje či neplánuje. Jak již bylo výše zmíněno, obě tyto činnosti samozřejmě probíhají. Na rozdíl od tradičních metodik však agilní přístup využívá kratších iterací, kdy jsou jednotlivé funkční celky dodávány cca ve 2 až 4 týdenních intervalech. Zpracovávané požadavky se řadí podle priority, kterou určuje zákazník. Hlavním určujícím parametrem při řazení je cena a časová náročnost odhadovaná na základě předpokládané obtížnosti. Za takto zadaných podmínek je pak úkolem

vývojového týmu dodat během jednotlivých iterací co nejvíce implementovaných požadavků. (Roudenský, 2013)

Zřejmě nejznámější agilní metodiku představuje Scrum. Byla vytvořena v devadesátých letech Kenem Schwaberem a Jeffem Sutherlandem.

V této metodice jsou dvě hlavní skupiny nazývané Pigs (prasata) a Chickens (kuřata). Terminologie vychází z povídky o praseti a kuřeti. Na vznikajícím projektu se podílejí oba - prase vkládá maso, kuře vkládá vejce. Rozdíl mezi nimi je v tom, že zatímco prase je přímo „zapojeno“, kuřete se problém „pouze týká“.

Analogicky jsou tak do skupiny Pigs zařazeny osoby, které přímo souvisejí s vývojem aplikace:

- *Product owner* (osoba zodpovídající za priority, za to, co se bude dít v dalším období, určující implementační detaily),
- *Scrum master* (řídí vývojáře, stará se o funkčnost počítačů, dostupnost software, řeší spory, atd.).

A do skupiny Chickens spadají uživatelé produktu, manažeři, kteří přímo nezodpovídají za vývoj:

- *Stakeholders* (lidé ze strany zákazníka, testeři, připomínky z venčí),
- *Managers* (osoby pomáhající s nastavením prostředí, ale nepatřící do kategorie vývojářů, Product ownerů nebo Scrum masterů).

Možnou, ale ne nezbytnou podmínkou tohoto typu metodiky je osobní přítomnost zákazníka při vývoji. Je tak možné průběžně pokládat otázky a ujasňovat potřebné detaily.

Při zahájení projektu je nejprve stanoven požadovaný výsledek. Následuje fáze Release Planning, kde je definován výčet vlastností a sepisují se User Stories (tj. případy použití systému).

Definované User Stories jsou sepsány do Product Backlogu. Úkoly se obodují podle časové náročnosti. Pro první iteraci neboli sprint, programátoři odhadnou, kolik úkolů by mohli za stanovený čas stihnout. Termín dokončení úkolů je označen jako Time box. Snahou je odhad stanovit tak, aby byl čas přesně využit, tedy se nepočítá s rezervami ani s rychlejším vypracováním.

Klíčovou podmínkou Scrum metodiky je skutečnost, že v naplánovaném sprintu již není povoleno dělat žádné změny. Vývojáři se tak mohou plně soustředit na zadané úkoly a nejsou ničím rozptylováni. Také je zakázána práce přesčas, aby byl vývojář dostatečně odpočinutý a další den pracoval s čistou hlavou a plně soustředěný.

Po dokončení sprintu se produkt předvede klientovi. V případě, že se nedodržel plán a některé úkoly nebyly dokončeny, vidí klient pouze hotovou část práce. Na základě předvedených výsledků a znalosti dokončení/nedokončení prací daného sprintu klient rozhoduje, zda se dokončí a jak se bude dále pokračovat. (Knesl, 2009b)

2.3 Testování ad hoc a exploratorní testování

Testování ad hoc představuje nesystematicky prováděnou aktivitu. Testy probíhají náhodně a bez plánování s cílem objevit co nejvíce defektů. Obvykle během tohoto typu testování není pořizována žádná dokumentace a pokrytí aplikace je reálně nízké.

Z těchto důvodů je jako ad hoc obecně označováno jakékoliv neefektivní testování bez stanovené strategie prováděné nekvalifikovanými testery. Ve skutečnosti však není úspěšnost tohoto testování zanedbatelná a může být vhodným doplňkem k formálnějším testovacím technikám. (Roudenský, 2013)

Náhodné testování nachází své uplatnění buď na začátku, ještě před spuštěním samotných testů, nebo také na konci procesu testování. Před začátkem testování se často jedná o prvotní testy, které ověří, že aplikace je vhodná k testování. Může zde dojít k zachycení prvních závažných chyb, které by jinak vedly k nemožnosti provedení testů. Na konci testování, kdy již proběhly všechny plánované testy a zbývající čas již není dostatečně dlouhý k provedení dalšího kola testů, mohou náhodné testy posloužit jako jistá forma regresního testu. (Anon., 2018b)

Exploratorní testování je možno chápat jako systematicky a důkladně pojaté ad hoc testování. Tester zkoumá produkt a na základ poznaného navrhuje a současně s tím provádí testovací případy. Zkoumání produktu zahrnuje zmapování jeho funkcí, dat a identifikaci potenciálně problémových oblastí, což do značné míry závisí na zkušenostech a znalostech nejen produktu, ale také jeho uživatelů nebo použitých technologií.

Tento termín byl poprvé použit v knize Testing Computer Software na konci osmdesátých let. Autorem je Cem Karner, jež je jedním ze zakladatelů přístupu testování řízeného kontextem.

Zkoumáním produktu se tester učí, jak jej používat. Na základě toho pak dokáže navrhnout strategii pro vyhodnocení jeho chování za určitých podmínek, tj. testovací případ. Ten ihned provádí a podle obdržených výsledků se buď dále soustředí na danou oblast, do které proniká hlouběji, anebo pokračuje dále.

Oproti testům založeným na specifikaci jde o interaktivní proces, kdy dynamická tvorba a provádění testovacích případů často vede k rychlému odhalení problémových částí.

Na rozdíl od ad hoc testování je exploratorní testování spíše, než na pouhé hledání defektů, zaměřeno více na poskytnutí důvěry v produkt a jeho výstupem by měl být nejen seznam defektů, ale také prověřené oblasti systému a nástin jejich testování. I přesto, že jde o formálnější způsob testování, než jakým je ad hoc, stále nelze exploratorní testování považovat za rovnocennou alternativu k testům podle specifikace. (Roudenský, 2013)

3 Metody a principy testování

Během vývoje produktu dochází k testování na mnoha různých úrovních a posléze také v různých testovacích prostředích. Testovány mohou být jednotlivé prvky, určité části či oblasti a také produkt jako celek.

Dříve než se budeme zabývat konkrétními metodami, které je pro testování možno využívat, uvedeme si obecně známé úrovně testování. Ačkoliv si pochopitelně každý vývojový tým provádí uzpůsobení dle svých potřeb, základní principy zůstávají zachovány a vycházejí z tohoto obecného nástinu.

3.1 Úrovně testování

Prvotní testy jednotlivých částí aplikace obvykle provádějí přímo sami programátoři. Jedná se o tzv. unit testy, které můžeme chápat jako ověřování jednotlivých stavebních bloků programu. Jednotky (units) jsou nejmenší testovatelné součásti. Cílem testu je ověřit každou jednotku nezávisle na ostatních a prokázat, že její chování je správné. (Roudenský, 2013)

Z jednotek se následně skládají moduly, které chápeme jako základní bloky, jejichž vzájemným propojením a interakcí vzniká systém. Modulové testování je obvykle nejnižší úroveň prováděná testovacím týmem. Je-li výsledný produkt tvořen prvky od více vývojových týmů různých společností, je úkolem modulového testování prokázat, že produkt je připraven na integraci do stabilního systému jako celku.

Integraci systému provádíme přidáváním dalších již otestovaných modulů do sestavy tak, že se začlení, začnou správně fungovat a nijak nenaruší stabilitu rozšiřovaného funkčního celku. Účelem integračních testů je odhalovat chyby vznikající při spojení a komunikaci jednotlivých prvků.

Výsledkem úspěšných integračních testů je stabilní systém připravený na systémové testování. V této fázi se ověřuje, zda systém splňuje požadavky specifikované zákazníkem a zároveň se jedná o poslední možnost jak odchytit a opravit defekty dříve, než na ně přijde zákazník. Na této úrovni se nejedná pouze o funkční testy, ale také o ověřování mimofunkčních požadavků. Následující výčet uvádí příklad prováděných testů (nejedná se o kompletní seznam)(Roudenský, 2013):

- *Funkční testy* - zda systém splňuje specifikované či implicitní požadavky,
    - *Bezpečnostní testy* - zda jsou data chráněna proti neoprávněnému přístupu zvenčí i zevnitř, jsou správná a neporušená, ověření autentizace, autorizace a dostupnosti dat i systému jako celku,
- *Testy robustnosti* - jak se systém dokáže vypořádat s chybami a selháními pokud nastanou,
- *Testy použitelnosti* - zaučení do systému a jeho použití je pro uživatele dostatečně komfortní a srozumitelné,
- *Testy interoperability* - systém či komponenta správně komunikuje s ostatními systémy (jedná se pouze o test komunikace, nikoliv zpracování převzatých dat),
- *Testy spolehlivost* - jak dlouho je systém schopen pracovat bez selhání,
- *Výkonnostní testy* - výkon systému sledovaný pomocí zvolených ukazatelů během různých scénářů,

- *Zátěžové testy* - simulace dlouhodobé zátěže pro ověření stability a použitelnosti,
- *Testy hraniční zátěže* - ověření chodu systému během hraniční zátěže, zjišťují se omezení systému a jaké je jeho chování, překračuje-li zátěž hranici,
- *Testy škálovatelnosti* - především u systémů, kde se očekává či plánuje nárůst zátěže.

Akceptační testování zákazníkem neboli UAT (User Acceptance Testing) má za úkol ověřit, že dodaný produkt splňuje tzv. akceptační kritéria, která byla mezi oběma stranami předem dohodnuta. Nejsou-li kritéria naplněna v očekávaném rozsahu, případně pokud jsou při akceptačních testech odhaleny zásadní chyby či nesrovnalosti, může dojít až k odmítnutí produktu.

Specifickou částí vývoje produktu je regresní testování. Jeho účelem je ověřit, že nedotčené části produktu (např. pokud doplňujeme určitou funkcionalitu do již odevzdaného systému) se po úpravě chovají stále stejně a nebyl do nich zaveden defekt. Regresní testování je třeba provádět na všech testovacích úrovních (jednotkově, modulově, integračně i systémově).

3.2 Statické testy

Statické testování probíhá na softwarovém produktu, aniž by byl výsledný software spuštěn, v extrémních případech může dojít k situaci, že dosud ani neexistuje. Typickým příkladem statického testování je testování jednotlivých dokumentů vzniklých v projektu a zdrojového kódu aplikace. Ačkoliv se může zdát tento druh testů zbytečný, může ušetřit nejen mnoho času, ale i finančních prostředků.

3.2.1 Testování dokumentace

Dokumentací se v tomto případě rozumí veškeré dokumenty, které jsou spojené s realizací daného produktu.

Testování obsahu dokumentů se zaměřuje především na správnost, úplnost a konzistenci obsahu a to nejen v rámci dokumentu, ale i mezi dokumenty navzájem. Jelikož cílem každého dokumentu je předat informace čtenáři, je tomuto účelu třeba primárně přizpůsobit jeho obsah. Hlavní kritéria, která by měl obsah dokumentu splňovat (Bureš, 2016):

- *Úplnost* - dokument obsahuje veškeré informace odpovídající charakteru dokumentu včetně odkazů na zdroje.
- *Správnost* - veškeré informace v dokumentu jsou správné/pravdivé.
- *Relevantnost* - jsou obsaženy pouze informace týkající se cíle projektu a účelu dokumentu.
- *Jednoznačnost* - formulace jsou voleny tak, aby nebylo možné text interpretovat více způsoby.
- *Konzistence* - text musí být konzistentní nejen v rámci dokumentu, ale i celého projektu.

Testování formy dokumentu zajišťuje, že bude výsledný text mít správnou strukturu obsahu, vzhled a nebude obsahovat gramatické či hrubé chyby. Ačkoliv je pravdou, že i formálně nedokonalý dokument může obsahovat zásadní informace pro realizaci, správná forma napomáhá orientaci v obsahu. A pokud je dokumentace jedním z hlavních výstupů projektu, nebo pokud je určena externím spolupracovníkům či zákazníkovi, může mít zanedbání formální stránky výrazné následky.

Samotné testování dokumentace pak můžeme rozčlenit do tří fází. V první fázi ověřujeme, zda byly dodrženy určené projektové šablony. V druhé fázi opravujeme zásadní a hrubé chyby - překlepy, gramatické chyby,… Třetí fáze představuje faktickou kontrolu. Jelikož při dlouhodobější práci s dokumentem může dojít k tzv. provozní slepotě (autor textu je již vůči jistým chybám, chybějícím významům a nedokončeným myšlenkám imunní), je vhodné využít alespoň jednu z následujících metod (Bureš, 2016):

- *Neformální revize* - metoda čtyř očí, text je nezávisle přečten jiným spolupracovníkem.
- *Walktrough* - připomínkovací prezentace dokumentu skupině spolupracovníků.
- *Technická revize* - poloformální revize dokumentu podle stanovených pravidel společnosti.
- *Inspekce* - oficiální kontrola skupinou dostatečně zkušených spolupracovníků, kteří dokument dostali v dostatečném předstihu k prostudování, na formálním setkání probíhá diskuze o připomínkách vedená moderátorem, který není autorem dokumentu.

3.2.2 Testování zdrojového kódu

V rámci testování zdrojového kódu rozlišujeme testy obsahu a formy. Při testování obsahu se zaměřujeme na robustnost, odolnost a efektivnost kódu. V kontrole formy sledujeme přehlednost a čistotu kódu. (Bureš, 2016)

Abychom mohli provést smysluplnou analýzu kódu, je třeba mít předem definováno, čeho chceme při psaní kódu dosáhnout - jaké charakteristiky obsahu jsou prioritní a očekávané formální náležitosti.

Jelikož kvalita kódu není jednoznačně definovatelná, je vždy hodnocení kódu vztahováno ke konkrétním žádaným cílům, nebo naopak k jevům, které jsou považovány za nežádoucí. Žádané cíle a nechtěné jevy v hodnocení kvality kódu (Bureš, 2016):

- ✓ *Udržitelnost a rozšiřitelnost* - kód má být dobře čitelný a srozumitelný, aby bylo možné ho v budoucnu rozvíjet a měnit (a to i v případě výměny programátorů).
- ✓ *Výkonová optimálnost* - podmínky, ve kterých se bude aplikace používat, vyžadují maximální rychlost nebo optimální práci s pamětí, kód musí být těmto požadavkům přizpůsoben, což může být někdy v rozporu s předchozím požadavkem.
- ✓ *Testovatelnost* - při psaní kódu je třeba myslet na to, aby se kód nebránil testování, je třeba volit vhodnou architekturu aplikace i organizace jednotlivých kusů kódu, abychom si při testování usnadnili práci, a aby bylo testování vůbec proveditelné.

- *Nevhodný návrh* - kód není srozumitelný, je těžké ho testovat například kvůli nemožnosti efektivně simulovat některé části kódu pro testovací účely.
- *Nesoulad s rozhraním podle specifikace* - kód dělá něco jiného, než říká specifikace, došlo k rozšíření rozhraní nad rámec specifikace.
- *Chybné použití globálních proměnných* - způsobuje chyby, u nichž je těžké identifikovat příčinu, hledáním a odstraněním takových chyb se tráví mnoho času.
- *Nedosažitelný kód* - kód, který nebude při běhu programu nikdy vykonán, zbytečně znepřehledňuje a zamlžuje zbytek kódu.
- *Inicializované, ale nevyužité proměnné nebo celé kusy nevyužitého kódu* - obvykle vznikají při úpravách existujícího kódu, kdy proměnná nebo metoda již není potřeba, ale definice je stále přítomna.

Chceme-li popsat kvalitu kódu exaktně a měřitelně, můžeme použít příslušné metriky. Zvolíme-li vhodnou metriku, můžeme na jejím základě odvozovat například počet a složitost testů, které bude třeba provést. V tomto ohledu je zajímavá například metrika cyklomatické složitosti. Ta určuje složitost kódu (zjednodušeně řečeno počet větvení kódu). Rozvětvený kód je nepřehledný, a také by bylo vhodné každé větvení otestovat, což vede k náročným a nákladným testům. Při použití vhodného návrhového vzoru, je možné dosáhnout zlepšení metriky. (Roudenský, 2016)

Podobným způsobem můžeme vyhodnocovat metriku provázanosti tříd. Ta vyjadřuje, jak moc jsou na sobě třídy kódu navzájem závislé - vysoká provázanost vede ke složitějšímu testování a složitější údržbě systému. (Bureš, 2016)

Abychom měli dobrý základ pro budoucí údržbu a rozšiřování kódu, optimalizujeme metriku pokrytí kódu testy. Dosáhneme-li 100 %, je každý řádek kódu vykonán v rámci nějakého testu. Ne vždy je však nutné dosáhnout tohoto stupně pokrytí, záleží na našem rozhodnutí, zda je třeba daný kus kódu testovat nebo nikoliv. (Bureš, 2016)

Pro samotné testování kódu se nejčastěji využívají tři hlavní metody (Bureš, 2016):

- *Revize kódu* - je podobá technické revizi dokumentace, spolupracovník čte a připomínkuje kód, revize by se měla provádět na dostatečně malých jednotkách, které již byly jednotkově otestovány a jsou připraveny k odeslání testovacímu týmu.
- *Párové programování* - na jednom úkolu pracují dva programátoři na společném počítači, zatímco jeden píše kód, druhý okamžitě poskytuje zpětnou vazbu, mohou tak vést dialog a ihned si ujasňovat, co má být implementováno.
- *Automatické testování kódu* - zakládá se na faktu, že kód musí být strojově interpretovatelný.

3.3 Dynamické testy

Na rozdíl od statického testování zkoumají dynamické testy chování a vlastnosti produktu za běhu. Dynamické testování je součástí většího celku nazývaného dynamická analýza. Pod tímto označením je shrnuto nejen testování ve všech jeho podobách, ale také využití různých nástrojů umožňujících například detailně analyzovat správu paměti a výkon systému, což může odhalit skryté a jinak těžko zjistitelné problémy. (Roudenský, 2013)

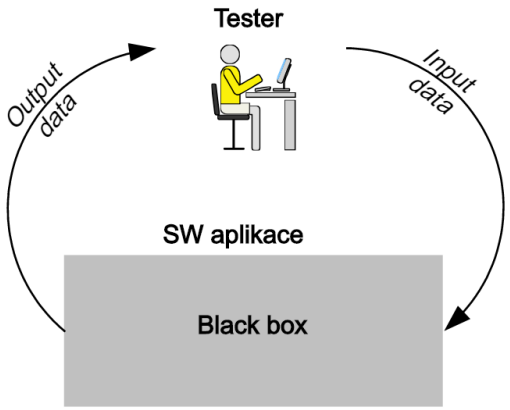
3.4 Testování metodou Smoke test

Jedná se o kontrolu stability systému a ověření, že všechny hlavní části projektu jsou po uvolnění nové verze aktivní a plně funkční. Výsledek testu rozhoduje o tom, zda bude zahájeno další testování. Jelikož smyslem tohoto testování není nacházení chyb, ale spíše testování připravenosti systému, jsou testovací případy postaveny jiným způsobem než u klasických testů. Jejich zadání je spíše obecné a povrchní ve smyslu - zadaná hodnota se vypíše na obrazovku, po zvolení položky v roletce nabídky se otevře patřičné okno atd. (Roudenský, 2013)

Provedení této kontroly by mělo předcházet započetí testování každé nové verze produktu. Pokud totiž již samo sestavení verze neproběhne v pořádku, může se při testech objevit velké množství chyb a to i v částech již dříve otestovaných a plně funkčních. Je-li tento problém odhalen ihned po uvolnění verze, předejde se časovým ztrátám při reportingu a přetestovaní hlášených chyb, neboť se rovnou zasáhne a provede vytvoření nové verze se správnou funkčností. (Borovcová, 2008)

3.5 Testování metodou Black box

Test černé skříňky se využívá k ověřování vstupů a výstupů, aniž bychom se zajímali o skutečnosti, které se dějí mezi tím. Probíhající procesy představuje právě zmíněná černá skříňka, která symbolizuje, že nevidíme a nevíme, co se děje uvnitř a jakým způsobem se z vloženého vstupu stal konečný výstup (viz Obrázek 9). Provedením tohoto typu testu obdržíme informaci, zda určené vstupy odpovídají požadovaným výstupům a obojí odpovídá funkční specifikaci.



Obrázek 9 - Testování metodou Black box (Steiner, 2005)

Někdy se testování metodou černé skříňky označuje jako test chování. Tím, že se nezajímáme o to, co se děje uvnitř produktu, ale pouze simulujeme situaci práce zákazníka s produktem, ověřujeme vlastně chování software při práci s ním.

Abychom mohli tento druh testů správně uskutečnit, je třeba mít dobře připravený podkladový dokument s požadavky nebo specifikací produktu. Není třeba vědět, co se má dít uvnitř skříňky, ale je třeba mít přesné zadání ve smyslu - zadáme-li hodnotu A na vstup, objeví se na výstupu hodnota B, provedeme-li pokyn C, bude následovat akce D, atd. (Patton, 2002)

Pokud všechny tyto informace máme shromážděny, můžeme na jejich základě vytvořit sadu testovacích případů, jejichž účelem je ověřit, že systém skutečně naplňuje veškeré očekávané chování. A to jak v kladném smyslu, tedy systém se chová podle zadaných předpokladů, tak i v negativním smyslu - zadáme-li nesmyslný nebo neočekávaný vstup/pokyn systém nahlásí chybu nebo se ukončí (i tyto situace by měli být v dobře provedené specifikaci produktu popsány).

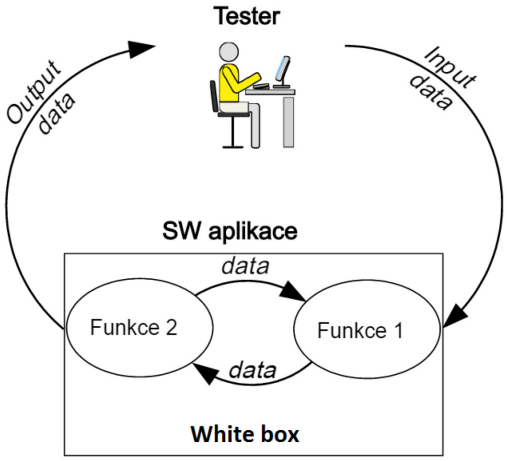
Výhody a nevýhody použití metody Black box (Čermák, 2010a):

- ✓ *Snadnost* - test může být prováděn i bez znalosti programovacích jazyků, operačních systémů.
- ✓ *Rychlost* - v krátkém období lze otestovat i rozsáhlé systémy.
- ✓ *Transparentnost* - test je srozumitelný pro zákazníka, chápe, co se bude testovat a jak, v případě akceptačních testů to může být on, kdo testovací scénáře vytváří a provádí.
- ✓ *Testovací scénáře mohou být napsány v okamžiku, kdy je kompletní specifikace.*
- ✓ *Testování není založeno na aktuální implementaci* - i když se změní programovací jazyk, operační systém a hardware, testování bude probíhat stejně, není nutné měnit testovací scénář.
- ✓ *Testerovi není nutné zpřístupňovat kód.*

- × *Nižší kvalita kódu* - to, že se na výstupu objeví očekávaná hodnota, neznamená, že je aplikace správně napsaná, kód může být značně neefektivní.
- × *Nežádoucí chování aplikace* - kromě požadované funkcionality může produkt provádět i jiné akce, které nejsou ve specifikaci a jejichž výstup se na standardním výstupu neobjeví, takže je test neodhalí.

3.6 Testování metodou White box

Metoda bílé skříňky funguje přesně obráceně než metoda Black box. A někdy se z tohoto důvodu také označuje jako Open box (otevřená skříňka). Zde se nezaměřujeme na vnější skutečnosti, ale sledujeme vnitřní algoritmy, jejich logiku a chování. Zkoumáme, zda jsou jednotlivé kroky prováděny relevantně, ve správném pořadí a navazují na ostatní činnosti tak, jak je očekáváno (viz Obrázek 10). Proto je tato metoda často užívána ve statickém testování.



Obrázek 10 - Testování metodou White box (Steiner, 2005)

Tato metoda klade na testovací tým značné nároky. Na základě poskytnuté dokumentace, binárního a zdrojového kódu testované oblasti se očekává, že bude tester schopen kódu porozumět a analyzovat ho. Někdy se také tento způsob testování nazývá audit zdrojového kódu.

Jedním z největších přínosů testování metodou White box je nalézání nežádoucího kódu. Obvyklé sady testů jsou totiž zaměřeny především na prokázání funkčnosti aplikace v požadované oblasti. Testování toho, zda aplikace neprovádí ještě něco navíc, je věnována minimální pozornost. Přitom právě nežádoucí kód může v systému způsobit velké obtíže, ať už je v produktu zapomenut omylem, jako pozůstatek, nebo implementován záměrně za účelem budoucího zneužití aplikace (například odesílání informací o uživatelích do schránky útočníka, povolení neautorizovaného přístupu při stisku určité klávesové zkratky, atd.). (Čermák, 2010b)

Výhody a nevýhody použití metody White box (Čermák, 2010b):

- ✓ *Včasné odhalování chyb* - analýza zdrojového kódu umožní odhalit chyby, kterých se programátor dopustil ještě dřív, než je kód zkompilován.
- ✓ *Odhalení nežádoucího kódu* - program může kromě požadovaných operací provádět i některé další nežádoucí operace, které mohou zůstat během jiných testů zcela nepovšimnuty.

- × *Náročnost* - vyžaduje výbornou znalost cílového systémů, testovacích nástrojů a programovacích jazyků.
- × *Vysoké náklady* - požaduje specializované nástroje jako jsou analyzátory zdrojového kódu, debuggery, atd.

3.7 Testování metodou Grey box

Šedá skříňka představuje kombinaci obou výše zmíněných metod dohromady. Můžeme tedy tento typ testování chápat tak, že zvolený konkrétní vstup sledujeme na cestě systémem až k výstupu a zkoumáme, zda se během průchodu chová tak, jak je očekáváno, aby dospěl k řádnému výsledku.

Na testera je kladen požadavek omezené znalosti interních datových a programových struktur. Jestliže tester ví, jak produkt funguje uvnitř, je pak lépe schopen ho otestovat zvenku (viz Obrázek 11).

Obrázek 11 - Testování metodou Grey box (Ramasubbarayalu, 2017)

Metoda Grey box se často užívá pro integrační testování dvou modulů od dvou různých dodavatelů, kdy je třeba otestovat jejich vzájemnou interakci a komunikaci. Také se často užívá při testování vícevrstvých aplikací, kdy kontrolujeme vstup i výstup a zároveň můžeme přistupovat do databáze. Jsme tedy schopni porovnávat všechny tři hodnoty a případně zjistit, v kterém místě dochází k manipulaci s výsledkem.

Výhody a nevýhody použití metody Grey box (Čermák, 2010c):

- ✓ *Slučuje výhody Black box i White box přístupu,*
- ✓ *Neintrusivní* - přístup ke zdrojovému ani binárnímu kódu není třeba, je založena na znalosti funkční specifikace, rozhraní a architektuře aplikace.
- ✓ *Inteligentní testy* - tester je schopen díky znalostem (byť omezeným) napsat inteligentní testovací scénáře zaměřené i na manipulaci s daty a použité komunikační protokoly.

- × *Neúplné otestování* - je dáno tím, že binární ani zdrojové kódy nejsou k dispozici a není tak možné otestovat všechny datové toky, míra pokrytí těchto toků závisí hodně na schopnostech, znalostech a zkušenostech testera.
- × *Kvalita kódu* - stejně jako u Black box testu to, že něco funguje podle specifikace a je to odolné proti známým zranitelnostem, neznamená, že je kód efektivní a že aplikace neobsahuje žádný nežádoucí kód.

3.8 Testování metodou End-to-end

Metoda často označovaná také jako E2E může být prováděna v různých variacích, ale vždy splňuje základní myšlenku - průchod sledovaného prvku po celou dobu jeho životnosti napříč systémem. Tedy od jednoho konce k druhému, jak již uvádí sám název metody. Během jeho cesty prověřujeme správnost chování a konečného stavu.

Tento typ testování se často používá v rámci scénářů pro akceptační testy. Simuluje se chování uživatele od prvního do posledního kroku se záměrem potvrdit správnost chování systému jako celku. Výsledkem testů je pak nejen znalost kvality jednotlivých součástí, ale i ověření, zda jsou podporovány potřeby uživatele.

3.9 Slovník

S ohledem na množství méně běžných a často z angličtiny přejatých termínů byl vytvořen slovník pojmů a zkratek vycházející z tohoto textu. Ačkoliv je zde vždy vysvětlení uvedeno přímo v kontextu, je možné si dohledat potřebné informace také

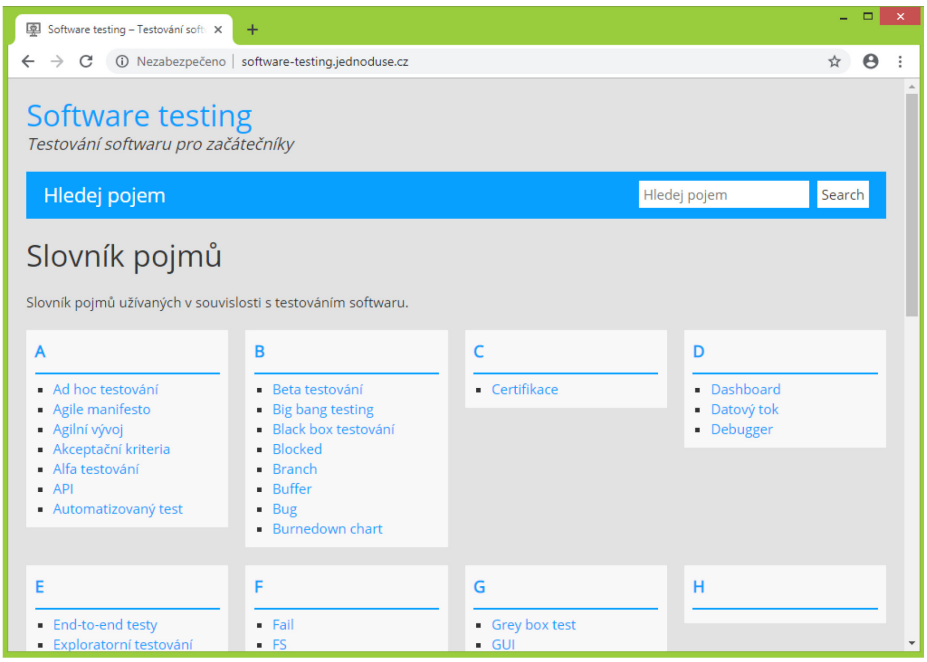
v příloze (viz 8 Příloha 1 - Slovník). Slovník obsahuje stručné vysvětlení pojmů, význam zkratek a jejich překlad (v případě nutnosti).

Veškeré pojmy lze také dohledat na webové adrese[1], kde byl slovník zveřejněn (viz Obrázek 12). Elektronická podoba umožňuje nadále rozšiřovat jak jeho rozsah, tak jednotlivé definice.

Pro zpracování webové stránky byl použit redakční systém WordPress.org a předzpracovaná šablona MyKnowledgeBase.

Aby bylo docíleno co největší přehlednosti a intuitivnosti ovládání, je stránka ponechána v co nejjednodušším vzhledu. Hlavním ovládacím prvkem je vyhledávací pole, které umožňuje na základě zadaných písmen, slova nebo celého termínu vyhledat všechna příslušná hesla.

Chceme-li procházet hesla bez vyhledávání, jsou pro větší přehlednost a snadnější orientaci seskupena pod hlavičky počátečních písmen. Po kliknutí na zvolený termín se otevře nová stránka s definicí. Chceme-li se seznámit se všemi hesly daného písmene, můžeme klepnout přímo na zvolené písmeno a na nově otevřené stránce se zobrazí všechna zařazená hesla a jejich definice.



Obrázek 12 - Náhled webu

[1] software-testing.jednoduse.cz

4 Automatizované testování software

Při vývoji nového projektu nebo rozšiřování funkcionality již vytvořené aplikace je třeba neustále ověřovat některé základní postupy a mechanismy. Jedná se o regresní testování, jehož podstata byla popsána v kapitole 3.1.

Pokud tyto testy provádí vývojář nebo tester, přichází tím o čas, který by mohl věnovat „důležitějším“ testům. V těchto případech je výhodné využívat nástroje pro automatické testování a automatizované testy. Jedná se o provedení stejných testů, jako by prováděl tester, jen jsou spuštěny hromadně a vyhodnoceny zcela automaticky.

Kromě tohoto zřejmě nejčastějšího využití, je automatizace testování využívána také pro následující případy (Roudenský, 2013):

- *Výkonnostní testování a jeho varianty* - simulace zátěže generované stovkami či tisíci uživateli kontinuálně po 24 hodin.
- *Funkční testy* - vhodně zvolená sada skriptů umožňuje rychle a efektivně otestovat všechny případy chování systému vyhodnocené jako relevantní, vysoce komplexní testy, případně takové, které vyžadují zvláštní znalosti a dovednosti je možné provádět (na základě správně vytvořeného skriptu) rychle, zcela spolehlivě a bezchybně.
- *Jednotkové testy* - testy základních prvků systémů bývají typicky (ne však výhradně) automatizovány.
- *Bezpečnostní testy* - v případě využití automatizace se obvykle jedná o ověřování odolnosti vůči známým způsobům napadení a jeho variantám, ne o inteligentní hledání slabin systémů.

Samozřejmě není možné testovací tým zcela nahradit pouze automatickými nástroji, vždy bude třeba lidského faktoru, který bude tyto nástroje spravovat, připravovat jednotlivé testovací scénáře a vyhodnocovat obdržené výsledky a statistiky. A pochopitelně je také třeba vzít v úvahu, že ne vždy je možné testy provádět automatizovaně. Některé skutečnosti je třeba vždy testovat manuálně.

Nejdůležitějšími vlastnostmi testovacích nástrojů a automatizace testů jsou (Patton, 2002):

- *Rychlost* - v porovnání s člověkem mohou automatické nástroje provádět testové případy několikanásobně rychleji.
- *Efektivita* - testovací nástroje šetří testerův čas, který může věnovat přípravě dalších testů, jejich plánování či provádění, je také možné využívat volný noční strojový čas.
- *Správnost a přesnost* - testovací nástroj neztrácí pozornost, nechybuje, jeho výkony jsou stabilní, vyrovnané a výsledná práce dokonalá.
- *Neúnavnost* - testovací nástroj nepociťuje únavu a tedy ani nevzdává uloženou práci, vykonává činnost, dokud nedospěje až do konce.

Nástroje pro automatické testování je možné v základu rozdělit na dva typy. První skupina nástrojů je označována jako neinvazivní. Tyto nástroje umožňují software monitorovat a sledovat, ale nedochází k žádným změnám parametrů ani jiným modifikacím. Můžeme tedy říci, že nás vlastně nezajímá to, co se děje uvnitř a jak nástroj

pracuje, ale pouze skutečnost, že nástroj dělá to, co dělat má. Z tohoto popisu vyplývá, že se jedná o nástroje pro testování metodou černé skříňky, která byla popsána v kapitole 3.5.

Druhá skupina nástrojů již vyžaduje určitou znalost principů a mechanismů, jež obsahují. Tento typ se nazývá invazivní a umožňuje provádět modifikace, manipulovat s operačním prostředím, apod. Podle zvoleného nástroje se míra invazivnosti liší a je vhodné si vybírat takové nástroje, které mají tuto míru co nejnižší. Možnost ovlivnění výsledků testů je tak udržována na co nejnižší míře. Vzhledem k popisu v kapitole 3.6 je zřejmé, že tento druh nástrojů je využíván k testování metodou bílé skříňky, neboť od testera vyžaduje pochopení procedur a znalost procesů, aby test mohl správně provést.

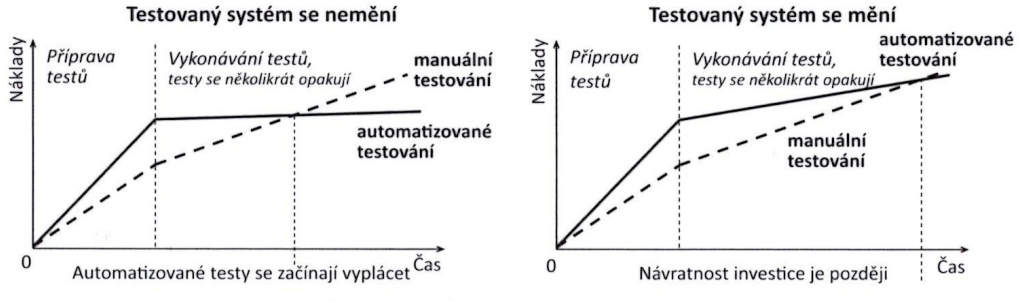
Pro provádění automatických testů jsou používány různé techniky, které se od sebe odlišují v mnoha různých parametrech. V praxi obvykle dochází k postupnému vytváření hybridních technik, tedy ke kombinování jednotlivých přístupů tak, aby bylo dosaženo žádaného výsledku. Obecně je možné techniky automatického testování rozdělit do čtyř základních oblastí, které je možné následně prolínat (Roudenský, 2013):

- *Zachycení a přehrávání aktivity uživatele* - jedná se o pravděpodobně nejznámější formu automatizace, testovací nástroj zachycuje aktivity uživatele provádějícího v rozhraní aplikace jednotlivé testovací případy a následně umožňuje jejich přehrávání, zjištění, zda test proběhl úspěšně je typicky založeno na porovnání určité proměnné a její očekávané hodnoty po skončení testu.
- *Modifikace vygenerovaných skriptů* - vycházíme ze zaznamenaných skriptů, ale pomocí znalosti skriptovacích jazyků jsme schopni v nich provádět úpravy, můžeme implementovat složitější logiku a rozšířit tak obsah testu, variabilitu proměnných a docílit tak lepší udržovatelnosti a znovupoužitelnosti skriptu.
- *Testování řízené daty* - využívá se v situacích, kde je třeba provést opakovaně testy využívající stejnou logiku, ale velký a variabilní objem vstupů a výstupů, ty jsou odděleny od samotného kódu pro provedení testu a jejich zdrojem může být např. databázová tabulka nebo soubor CSV či XLS, testovací skript načítá data řádek po řádku, provádí potřebné operace a porovnává získané výsledky s těmi očekávanými, které stejně jako vstupy načítá ze zdrojových dat.
- *Testování řízené klíčovými slovy* - je podobné předchozímu typu, ale zdrojové tabulky neobsahují jen vstupní data, ale i příkazy, ze kterých sestává testovací skript - klíčová slova, ta jsou za běhu načítána, prováděna a tak je dynamicky vytvářena samotná logika procedury testu. Jelikož klíčová slova představují obecné akce v uživatelském rozhraní (kliknutí na tlačítko, zadání hodnoty, atd.), může tester z klíčových slov a potřebných parametrů poskládat testovací případ a návrh testu je tak prakticky zcela oddělen od jeho nízko úrovňové implementace, čímž je dosaženo vysoké míry abstrakce.

Veškeré doposud uváděné informace byly zaměřeny na technickou stránku automatizovaného testování a shrnovaly jeho přínosy pro vývoj produktu. Zavedení automatizace však s sebou přináší i „stinné“ stránky. Ty se však dotýkají zejména

manažerské vrstvy organizace projektu, neboť se jedná především o zvýšené množství nákladů ve vstupním procesu.

Dříve než dojde k procesu zavádění automatizace, je třeba si udělat pečlivou a podrobnou rozvahu o výhodnosti a účelnosti tohoto typu testů. Na níže uvedeném obrázku (viz Obrázek 13 - vlevo) můžeme vidět, že prvotní náklady pro zavádění automatizovaných testů výrazně převyšují náklady na manuální testování. Tato skutečnost vyvstává nejen z výdajů na pořízení potřebného softwarového, případně i hardwarového vybavení, ale také z nákladů na vyškolení zodpovědné osoby/osob a případně rozšíření testovacího (a/nebo vývojového) týmu o další osoby. Po zavedení automatických testů a ustálení jejich užívání je pak z grafu zřejmé, že náklady začnou být konstantní - skripty jsou napsány, pracovníci vyškoleni. Situaci tedy můžeme chápat tak, že (například v případě blížícího se dokončení produktu) se regresní testy provádí automaticky a šetří tedy čas testovacímu týmu, který se věnuje testům nových funkcionalit. Křivka nákladnosti manuálních testů však stále stoupá, protože (ve stejné situaci jako u automatizovaného testování) se zvětšují balíky nutných regresních testů při každé dokončené změně a zároveň se dokončují průběžné testy nových funkcionalit.



Obrázek 13 - Návratnost investice do automatizace v čase (Bureš, 2016)

Podíváme-li se na situaci v okamžiku, kdy začneme do produktu vnášet zásadní změny (viz Obrázek 13 - vpravo), opět vidíme, že nákladnost automatizovaných testů je vyšší než u manuálních. To vyplývá ze skutečnosti, že je třeba věnovat důkladnou péči všem automatizovaným skriptům a aktualizovat je pro prováděné změny. Tato činnost je podstatně náročnější než úpravy v běžných scénářích pro manuální testování a tedy i nákladnější.

Shrnutí výhod a nevýhod automatického testování:

- ✓ *Rychlost,*
- ✓ *Přesnost,*
- ✓ *Bezchybnost,*
- ✓ *Neúnavnost,*
- ✓ *Možnost využití času mimo pracovní dobu (dávkové spouštění).*

- × *Je třeba vyškolit alespoň jednoho člověka v novém software.*
- × *Je třeba mít precizně vypracované scénáře, jinak je do projektu vnášena chyba.*
- × *Je třeba scénáře neustále aktualizovat, aby odpovídaly nejnovější podobě projektu.*
- × *Využití je možné pouze v jednoznačně daných situacích.*

V konečném důsledku tedy můžeme říci, že je u zavádění automatického testování třeba počítat s vyššími vstupními náklady a pozdější návratností. Ale naučíme-li se nástroje plnohodnotně používat, vytvoříme kvalitní základy nejen pro projekt, na kterém aktuálně pracujeme, ale i pro všechny budoucí.

5 Skutečná podoba testování

V této části práce si předvedeme, jak vypadá testování v praxi a to jak v podobě manuální, tak automatizované. Dříve než přistoupíme k samotnému praktickému příkladu, je třeba si stanovit parametry, které budou u obou variant sledovány a následně hodnoceny a také využívané nástroje.

Jak již bylo zmíněno v kapitole 1.3 Aktivity vedoucí k plánování testování, je třeba najít vhodný poměr ceny, kvality a investovaného času. Ačkoliv se tato myšlenka v původním znění týká vývoje celého projektu, platí stejné pravidlo i pro vytváření a provádění testovacích scénářů. Abychom tedy mohli hodnotit výhody a nevýhody manuálního a automatizovaného testování budeme sledovat:

- celkový čas přípravy testu,
- celkový čas provedení testu,
- doba nutná pro dokumentaci nalezených chyb,
- nezbytné znalosti a schopnosti testera,
- časové možnosti provádění testů.

K názorné dokumentaci příprav a provádění testovacích scénářů budeme využívat níže uvedené nástroje.

Jira s nádstavbou Zephyr

Jedná se o nástroj pro evidenci chyb a správu testovacích scénářů od společnosti Atlassian. Nástroj podporuje projektové řízení, nastavitelné workflow pro jednotlivé typy úkolů/problémů, neustále dostupné informace pro celý tým přes webové rozhraní. Je využíván k rozepisování a provádění jednotlivých testovacích scénářů, vytváření a vyhodnocování reportů z jednotlivých testovacích běhů a celého průběhu projektu. Nástroj je variabilní a je možné si ho přizpůsobit podle požadavků pro daný pracovní tým i zpracovávaný projekt. (Atlassian, 2019)

V našem případě využijeme software Jira k vytvoření ukázky vzhledu a provádění testovacích scénářů. A to jak pro manuální, tak pro automatizované testování.

Jubula

Nástroj pro automatizované provádění testů grafického rozhraní. Je podporováno vytváření testů, jejich spouštění a následná analýza výsledků. Software lze využít pro testování Java, HTML a PHP aplikací. Vytváření scénářů probíhá prostřednictvím řazení jednotlivých příkazů (klik myší, vstup z klávesnice, ověření viditelnosti grafické komponenty, apod.) za sebe z předdefinované knihovny v požadovaném pořadí. Každý příkaz je možné detailněji specifikovat v podrobném nastavení. Pro provedení testu je také nutné definovat vlastní sadu vstupních a výstupních dat. (Eclipse Foundation, Inc., 2019)

Předvedeme si, jak vypadá test, připravený v nástroji Jubula.

Jenkins

Jedná se o volně dostupnou Java aplikaci s otevřeným kódem. Jeho autorem je Kohsuke Kawaguchi a byl vyvinut pod licencí MIT (Massachusetts Institute

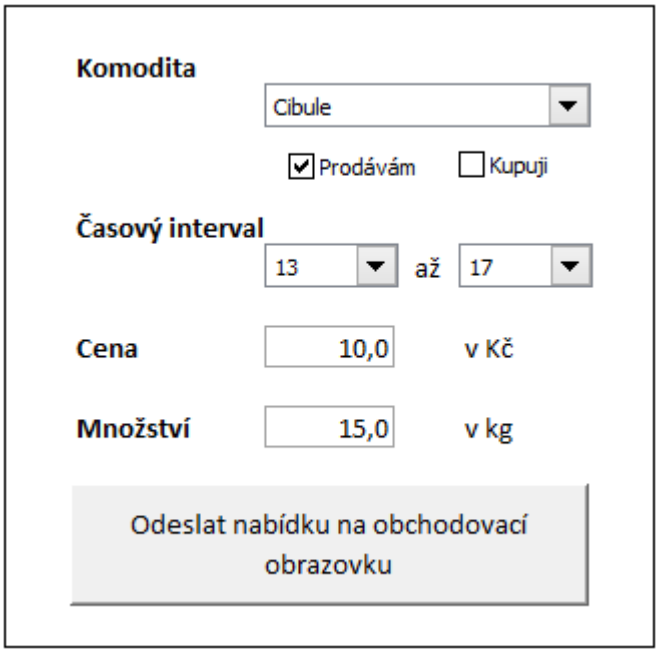
of Technology). Tento nástroj je kompletním serverovým systémem pro automatizaci technických procesů a činností souvisejících s vývojem, testováním a nasazením nebo poskytováním software. (Kawaguchi, 2019)

Pro naši ukázkovou situaci bude software Jenkins využit pouze na pozadí. U automatizovaných testů vzniklých v software Jubula je používán k jejich spouštění a propisování obdržených výsledků do testovacích scénářů napsaných v platformě Jira.

5.1 Příklad užití

Na základě teoretických informací uvedených v předcházejících kapitolách nyní rozebereme ukázkový příklad užití a jeho testování různými způsoby. Základem následujícího rozboru bude smyšlený software určený pro obchodování s různými komoditami. Uživatel aplikace může komodity prodávat či nakupovat. Systém v celém svém rozsahu umožňuje vkládání nabídek na prodej či nákup, jejich přehledné vystavení na obchodovací obrazovce a zároveň také různá vlastní nastavení ohledně limitů nabídek, upozornění na výhodné nabídky, přehledy provedených transakcí apod. Z celého systému bude pro nás podstatná pouze jedna část a to panel, ze kterého je nabídka zaváděna do systému a jeho chování. S tím souvisí také uživatelské nastavení limitů a jeho vliv na zavedení nabídky.

Obchodování probíhá v určených časových intervalech a zadavatel nabídky si vybírá v kterém časovém intervalu, za jakou cenu a v jakém množství chce svoji komoditu nabídnout či od jiného uživatele odkoupit. Nabídku zavadí prostřednictvím panelu (viz Obrázek 14), kde jednotlivé položky volí z roletkového menu anebo zadává z klávesnice. V případě, že nabídku zadá správně, je odeslána do systému a hodnoty propsány na obchodovací obrazovku, kde je uvidí ostatní uživatelé systému.



Obrázek 14 - Panel vložení nabídky

Pro řádné vyplnění jednotlivých položek jsou stanovena pravidla, která je nutno dodržet. Mimo tato systémová pravidla si může zadavatel ještě definovat osobní pravidla

týkající se minimálního/maximálního nákupního množství, minimálního/maximálního prodejního množství a minimální/maximální ceny (viz Obrázek 15). Tyto hodnoty si uživatel stanovuje pouze pro sebe a při zadání nabídky musí vždy být provedena kontrola, zda jsou limity nastaveny a pokud ano, je-li nabídka v rámci limitů.

komodita	cena		množství	
	min.	max.	min.	max.
Brambory		8	7	
Kukuřice	7	15		20
Cibule	9		10	50

Obrázek 15 - Nastavení limitů pro uživatele

Na rozdíl od systémových pravidel, která nelze obejít, je při překročení uživatelských hraničních hodnot zobrazeno hlášení s dotazem, zda nabídku skutečně odeslat nebo zda chce uživatel provést opravu překročené hodnoty.

Pro přesnější představu očekávaného chování panelu si nyní uvedeme postup zadání nabídky v bodech:

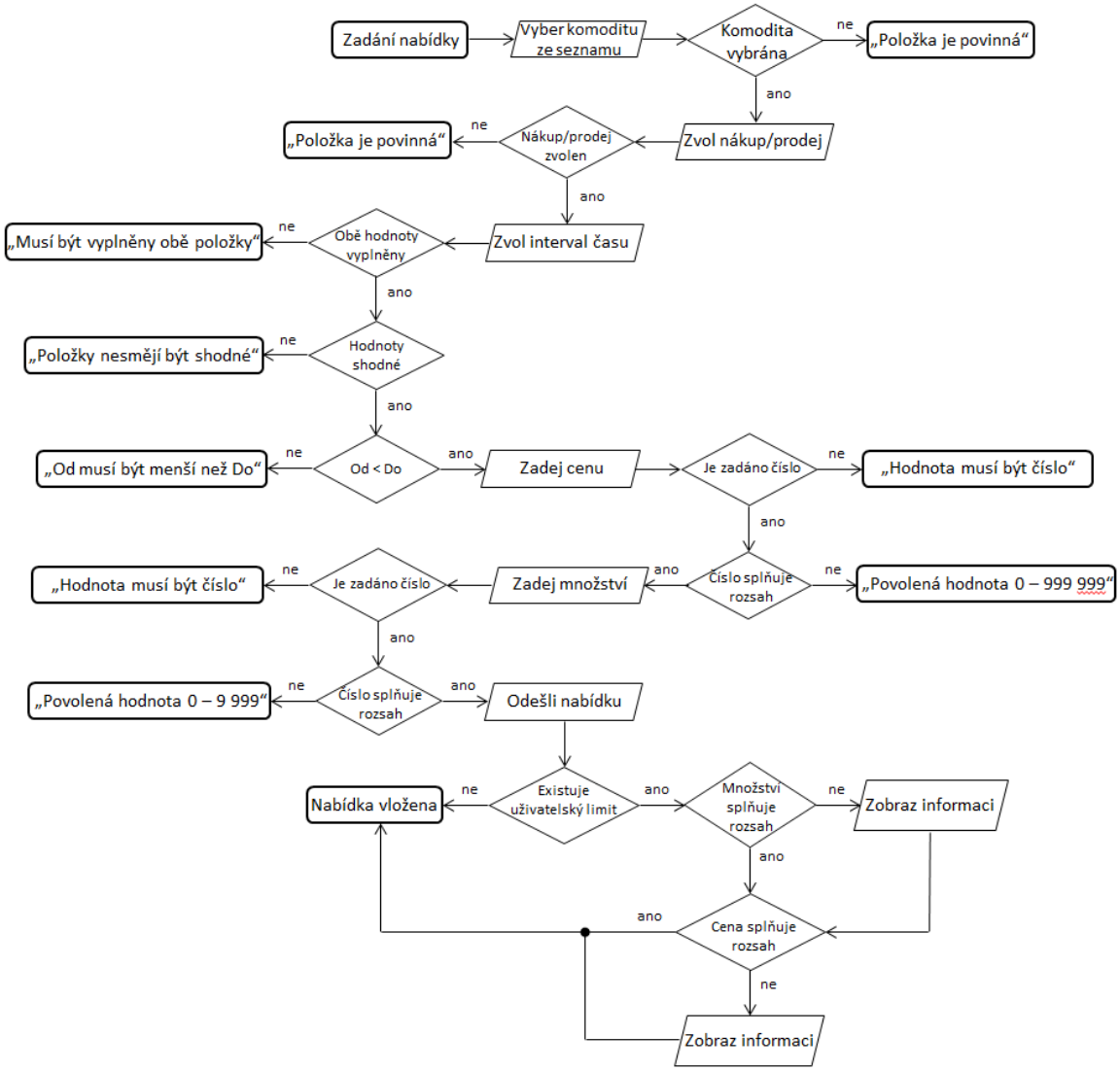
1) v roletkovém seznamu komodit zvolíme zboží,
2) zaškrtneme pole podle toho, jestli chceme zboží koupit či prodat,
3) zvolíme časový interval, po jehož dobu trvání bude naše nabídka viditelná na obchodovací obrazovce pro ostatní uživatele,
4) prostřednictvím klávesnice vepíšeme cenu, za kterou chceme zboží prodat/koupit,
5) prostřednictvím klávesnice vepíšeme množství, které hodláme prodat/koupit,
6) nabídku odešleme na obchodovací obrazovku stiskem tlačítka ve spodní části panelu.

Takto jednotlivé kroky vnímá uživatel systému, ve skutečnosti se ale odehrává ještě mnoho dalších kroků na pozadí. Uveďme si nyní podrobnější rozpis chování panelu včetně „neviditelných“ událostí:

1) v roletkovém seznamu komodit zvolíme nabízené/poptávané zboží; jelikož zde vybíráme z předem stanoveného seznamu, probíhá pouze kontrola, zda je položka vyplněna,
2) zaškrtneme pole podle toho, jestli chceme zboží koupit či prodat; je vykonána kontrola, že je zaškrtnuto právě jedno z dostupných polí,
3) zvolíme časový interval, po jehož dobu trvání bude naše nabídka viditelná na obchodovací obrazovce pro ostatní uživatele; kontroluje se, že zvolená čísla jsou od sebe různá a zároveň je první číslo menší než druhé,
4) prostřednictvím klávesnice vepíšeme cenu, za kterou chceme zboží prodat/koupit; je kontrolováno, že byla vložena číselná hodnota v systémem stanoveném rozsahu (např. 0 - 999 999),
5) prostřednictvím klávesnice vepíšeme množství, které hodláme prodat/koupit; je kontrolováno, že byla vložena číselná hodnota v systémem stanoveném rozsahu (např. 0 - 999),

6) nabídku odešleme na obchodovací obrazovku stiskem tlačítka ve spodní části panelu; při stisku tlačítka je vyhodnoceno, zda všechny předchozí kontroly proběhly úspěšně, pokud ano, je nabídka odeslána na obchodovací obrazovku, v případě, že je překročen některý z limitů stanovených uživatelem, je zobrazena informace o této skutečnosti a uživatel zvolí, zda nabídku přesto odeslat anebo se vrátí do panelu zavedení nabídky a hodnotu upraví.

V grafické podobě si můžeme celý proces představit jako větvený graf (viz Obrázek 16):



Obrázek 16 - Diagram průběhu zavedení nabídky

Z obrázku i slovního popisu je zřejmé, že i když se jedná pouze o jednu malou část celého systému, skrývá se zde poměrně velké množství potenciálních problémů, které je třeba prověřit.

Nejprve si tedy případ užití rozložíme na jednotlivé situace (test case):

- Vzhled:
  - Vzhled panelu zavedení nabídky odpovídá předpisu.
- Akce:
  - Tlačítka jsou funkční, na správných místech a se správným popisem.

- Procesy:
    - nejsou žádné uživatelem stanovené limity,
        - tabulka zavedení nabídky je správně vyplněna => nabídka je zavedena do systému,
        - tabulka není správně vyplněna => nabídka není zavedena do systému,
    - uživatel stanovil limity,
        - tabulka je správně vyplněna a hodnoty jsou v limitu => nabídka je zavedena do systému,
        - tabulka je správně vyplněna a hodnoty nejsou v limitu => nabídka není zavedena do systému (je zobrazena dotazovací hláška),
        - při zvolení opravy hodnoty => návrat na zadání
        - při zvolení odeslání nabídky => nabídka je zavedena

Následně tyto test case rozdělíme na jednotlivé testovací scénáře, kterými pokryjeme kontrolu vizuální stránky i všechny možné varianty kontroly průchodu systémem. Procesní scénáře budou ověřovat jak pozitivní, tak hraniční i chybové situace. Abychom jednotlivé typy scénářů odlišili, použijeme identifikační označení pomocí zkratek:

- FV - formulář vzhled,
- FC - formulář akce,
- PZ - základní scénář (průchod základní),
- PA - alternativní chování systému (průchod alternativní),
- PE - simulace chybového chování (průchod chybový - errorový).

V následující tabulce (viz Tabulka 2) je shrnuta sada testovacích scénářů na základě výše stanovených kritérií. Názvy scénářů jsou záměrně voleny velmi popisně, aby bylo zřejmé, kterou situaci pokrývají. V praxi jsou pochopitelně užívány názvy kratší, například s číselným odkazem na kapitolu funkční specifikace věnující se dané oblasti, a pouze s heslovitým označením testované situace.

Tabulka 2 - Rozpad testovacích případů na scénáře

Test case	Test		Název
1	**Vzhled panelu zavedení nabídky odpovídá předpisu**		
	1.1.	FV	Vzhled panelu - uspořádání polí, popisky, zarovnání, umístění tlačítek odpovídají předpisu
2	**Tlačítka jsou funkční, na správných místech a se správným popisem**		
	2.1.	FC	Aktivní prvky - veškerá tlačítka jsou aktivní/neaktivní v očekávané situaci, reagují řádně na klepnutí, provádí očekávanou akci
3	**Bez uživatelských limitů, tabulka zavedení nabídky je správně vyplněna => nabídka je zavedena do systému**		
	3.1.	PZ	Zavedení nabídky - všechny údaje řádně vyplněny, po správném vyplnění všech údajů se zpřístupní tlačítko odeslání nabídky

Test case	Test		Název
4	**Bez uživatelských limitů, tabulka není správně vyplněna => nabídka není zavedena do systému**		
	4.1.	PE	Zavedení nabídky - jednotlivé údaje nezvoleny, zavedeny mimo povolené rozsahy hodnot, při každém porušení pravidla vyplnění je pro danou položku zobrazena chybová hláška, tlačítko odeslání nabídky není dostupné (zpřístupní se pouze při správném vyplnění formuláře)
5	**Stanoveny uživatelské limity, tabulka zavedení nabídky je správně vyplněna => nabídka je zavedena do systému**		
	5.1.	PZ	Zavedení nabídky - všechny údaje řádně vyplněny, číselné hodnoty jsou v mezích uživatelsky stanovených limitů, nabídka je odeslána do systému bez zobrazení chybové hlášky
	5.2.	PA	Zavedení nabídky - všechny údaje řádně vyplněny, číselné hodnoty jsou nastaveny jako horní hraniční hodnoty uživatelsky stanovených limitů, nabídka je odeslána do systému bez zobrazení informační hlášky
	5.3.	PA	Zavedení nabídky - všechny údaje řádně vyplněny, číselné hodnoty jsou nastaveny jako dolní hraniční hodnoty uživatelsky stanovených limitů, nabídka je odeslána do systému bez zobrazení informační hlášky
6	**Stanoveny uživatelské limity, tabulka není správně vyplněna => nabídka není zavedena do systému**		
	6.1.	PE	Zavedení nabídky - všechny údaje řádně vyplněny, číselné hodnoty jsou nastaveny pod dolní hraniční hodnotou uživatelsky stanovených limitů, při odeslání do systému je zobrazena informační hláška o překročení limitu s dotazem na potvrzení odeslání, dotaz je potvrzen a nabídka odeslána.
	6.2.	PE	Zavedení nabídky - všechny údaje řádně vyplněny, číselné hodnoty jsou nastaveny nad horní hraniční hodnotou uživatelsky stanovených limitů, při odeslání do systému je zobrazena informační hláška o překročení limitu s dotazem na potvrzení odeslání, dotaz je zamítnut, uživatel je vrácen na panel zavedení nabídky.

Můžeme si všimnout, že při testování hraničních hodnot a jejich překročení, jsou vytvořeny zvláštní scénáře pro hraniční minimum/maximum a jeho překročení. Obojí by mohlo být pochopitelně simulováno v rámci jednoho scénáře, ale tímto rozpadem se zjednoduší administrativa okolo plánování scénářů a následných retestů. V případě, že bude problematickým shledána pouze jedna strana intervalu, je možné po opravě naplánovat pouze ten scénář, kterého se chyba přímo týkala. Tedy testování provést efektivně v co nejkratším čase.

Z tabulky je zřejmé, že prvotní testy jsou zaměřeny na čistě vizuální podobu formuláře a jeho aktivní prvky. Tyto testy se obvykle provádějí při úplně prvním uvolnění nové části projektu. Vizuální provedení se porovná s funkční specifikací, tester „dostane do oka“, jak má formulář vypadat a že skutečně reaguje, jak má. V následujících bězích, pokud nedojde k zásadní změně vizuální podoby, se již tyto scénáře znovu nepoužívají a veškerá pozornost se soustředí na procesy a chování systému. Dojde-li tedy z jakékoliv příčiny k odchylce ve vzhledu, reportuje tuto skutečnost tester i mimo scénáře přímo určené k vizuální kontrole.

Naše pozornost bude tedy zaměřena především na procesní scénáře. Na nich si budeme demonstrovat jednotlivé typy testů a jejich případnou vhodnost/nevhodnost pro automatické testování.

5.2 Black box test

Využití tohoto typu testu je vhodné jak pro Smoke test v okamžiku, kdy komponentu obdržíme k testování, a chceme ověřit, že testy mohou začít (viz 3.4 Testování metodou Smoke test), tak pro následné zařazení do sady UAT testů. Můžeme totiž přirozeně předpokládat, že jediný zájem, který zákazník bude u této části systému mít je, že při splnění všech podmínek se nabídka řádně zadá. Vedlejší skutečnosti podílející se na tom, že k tomu skutečně dojde, nejsou pro běžného uživatele podstatné.

Pro testování Black box se tedy jeví jako vhodné využít scénáře:

- **3.1. PZ** *Zavedení nabídky - všechny údaje řádně vyplněny, po správném vyplnění všech údajů se zpřístupní tlačítko odeslání nabídky,*
- **4.1. PE** *Zavedení nabídky - jednotlivé údaje nezvoleny, zavedeny mimo povolené rozsahy hodnot, při každém porušení pravidla vyplnění je pro danou položku zobrazena chybová hláška, tlačítko odeslání nabídky není dostupné (zpřístupní se pouze při správném vyplnění formuláře),*
- **5.1. PZ** *Zavedení nabídky - všechny údaje řádně vyplněny, číselné hodnoty jsou v mezích uživatelsky stanovených limitů, nabídka je odeslána do systému bez zobrazení informační hlášky,*
- **6.1. PE** *Zavedení nabídky - všechny údaje řádně vyplněny, číselné hodnoty jsou nastaveny pod dolní hraniční hodnotou uživatelsky stanovených limitů, při odeslání do systému je zobrazena informační hláška o překročení limitu s dotazem na potvrzení odeslání, dotaz je potvrzen a nabídka odeslána,*
- **6.2. PE** *Zavedení nabídky - všechny údaje řádně vyplněny, číselné hodnoty jsou nastaveny nad horní hraniční hodnotou uživatelsky stanovených limitů, při odeslání do systému je zobrazena informační hláška o překročení limitu s dotazem na potvrzení odeslání, dotaz je zamítnut, uživatel je vrácen na panel zavedení nabídky.*

Jak názvy jednotlivých scénářů napovídají, bude jejich struktura ve všech případech podobná. Tester se přihlásí do aplikace jako uživatel a provede sadu kroků, jejichž výsledkem bude zavedení/nezavedení nabídky na obchodovací obrazovku. Abychom si dokázali lépe představit průběh testu, rozepíšeme si scénář 6.1. PE na jednotlivé testovací kroky (viz Obrázek 17).

Rozepsání scénáře pro provedení Black box testu je časově nejméně náročné a neobsahuje velké množství kroků. Ani při provádění nejsou na testera kladeny nijak komplikované nároky a lze předpokládat, že testování netrvá v ideálním bezchybném případě déle než několik minut.

Tento typ scénářů je vhodné zadávat k provedení novým testerům, aby se seznámili se způsobem formulace pokynů ve scénářích, s logikou jejich provádění a vyplňování.

6.1 PE Zavedení nabídky - překročena dolní uživatelská hranice

Popis:
- všechny údaje řádně vyplněny
- číselné hodnoty jsou nastaveny pod dolní hraniční hodnotou uživatelsky stanovených limitů
- při odeslání do systému je zobrazena info hláška o překročení limitu s dotazem na potvrzení odeslání
- dotaz je potvrzen a nabídka odeslána

	Testovací krok	Testovací data	Očekávaný výsledek	Vykonáno	Komentář testera
1.	Přihlaste se do aplikace jako běžný uživatel.	např. Uživatel 01	Uživatel úspěšně přihlášen	PASS	*Přihlášen Uživatel 01*
2.	Nastavte limit minimální ceny a množství pro komoditu " Cibule".	*Nastavení -> Uživatelské limity* Komodita: Cibule Min.cena: 10 Min. množství: 5 tlačítko *Uložit*	Hodnoty minimálního množství a ceny pro komoditu "Cibule" nastaveny.	PASS	*Limit nastaven podle popisu*
3.	Spusťte Obchodovací obrazovku.	*Obchodovací obrazovka*	Obchodovací obrazovka spuštěna.	PASS	
4.	Zaveďte do systému nabídku na prodej cibule.	*Komodita* : Cibule *Prodej* *Čas. int.* : 13-15 *Cena* : 6 *Množství* : 6 Odeslat	V panelu zavedení nabídky jsou vyplněny předepsané hodnoty. Při odeslání nabídky do systému stiskem tlačítka je zobrazeno hlášení: *"Překročen limit minimální ceny, skutečně chcete nabídku odeslat?"*	PASS	*Panel zavedení nabídky vyplněn podle popisu. Informační hláška zobrazena.*
5.	V zobrazeném dotazu stiskněte "*NE*" a upravte hodnoty zadávané nabídky.	tlačítko *NE* *Komodita* : Cibule *Nákup* *Čas. int.* : 13-15 *Cena* : 12 *Množství* : 3 Odeslat	V panelu zavedení nabídky jsou vyplněny předepsané hodnoty. Při odeslání nabídky do systému stiskem tlačítka je zobrazeno hlášení: *"Překročen limit minimálního množství, skutečně chcete nabídku odeslat?"*	FAIL	*Panel zavedení nabídky vyplněn podle popisu. Po odeslání nabídky do systému nebyla zobrazena inf. hláška a nabídka se odeslala*
6.	V zobrazeném dotazu stiskněte "*Ano*" a ověřte, že se nabídka zavedla do systému.	tlačítko *ANO*	Nabídka byla zavedena do systému a ve stanoveném intervalu se zobrazuje na obchodovací obrazovce.	BLOCK	
7.	Na Obchodovací obrazovce zkontrolujte zavedenou nabídku a zobrazené hodnoty		Nabídka je zavedena na obchodovací obrazovce. Data vyplněná v jednotlivých sloupcích odpovídají zadání a jsou uvedena ve správném tvaru	UNEXECUTED	

Obrázek 17 - Testovací scénář Black box

Jak můžeme na uvedeném příkladu vidět, tester nalezl v kroku 5 chybu. Nabídka byla do systému zavedena, ačkoliv bylo očekáváno jiné chování. Krok je proto označen jako FAIL a následující jako BLOCK, neboť ho nebylo možno provést. Krok 7 je ponechán v původním stavu, tedy nebylo k němu vůbec přikročeno. Jakým způsobem jsou značeny kroky po té, co je nalezena během testovacího scénáře chyba, je samozřejmě čistě na domluvě daného testovacího týmu. Všechny kroky za chybným s označením FAIL je možné nechat jako neprovedené, což značí, že ve scénáři nebylo možno dále pokračovat. Je také možné označit všechny další kroky jako BLOCK, ve výsledku je význam stejný, ale může tím být znázorněno, že tester přečetl i další kroky a žádný z nich již není možné provést, neboť jsou všechny zablokovány nalezenou chybou (v našem případě by tedy měl

být i poslední krok označen BLOCK). Jedná-li se o rozsáhlejší test, je možné označit všechny ovlivněné kroky BLOCK a pokračovat v testu od místa, které již není nalezenou chybou ovlivněno a provést scénář až do konce (nebo do další chyby). V takovém případě se pak při ověřování provedené opravy často postupuje tak, že jsou z daného scénáře testovány jen kroky zasažené chybou a označené BLOCK. Opět se jedná o to, využívat testovací čas efektivně a věnovat pozornost tomu, co je skutečně důležité.

5.3 Grey box test

Jelikož testování Grey box již předpokládá jisté základní znalosti systému a jeho fungování (viz kapitola 3.7 Testování metodou Grey box), můžeme do této kategorie zařadit zbývající dva scénáře:

- **5.2. PA** *Zavedení nabídky - všechny údaje řádně vyplněny, číselné hodnoty jsou nastaveny jako horní hraniční hodnoty uživatelsky stanovených limitů, nabídka je odeslána do systému bez zobrazení informativní hlášky,*
- **5.3. PA** *Zavedení nabídky - všechny údaje řádně vyplněny, číselné hodnoty jsou nastaveny jako dolní hraniční hodnoty uživatelsky stanovených limitů, nabídka je odeslána do systému bez zobrazení informativní hlášky.*

Zatímco předchozí scénáře pouze naplňují chování funkcionality, které je intuitivně očekávané i neznalým uživatelem, tyto alternativní scénáře se zabývají hraničním stavem. Je zde třeba provádět pečlivější kontrolu a nabízí se rozšířit kroky scénáře o případnou kontrolu zavedení uživatelských limitních hodnot do tabulek, se kterými je posléze prováděno porovnání při odesílání nabídky do systému.

Budeme například uvažovat, že náš software umožňuje stanovit uživatelské limity pouze v celých číslech, ale hodnoty v nabídkách je možné zadávat s jedním desetinným místem (tato situace je poněkud absurdní, ale velmi ilustrativní). Zároveň také pracuje s běžným matematickým zaokrouhlováním, tedy do hodnoty 0,5 zaokrouhluje dolů a od hodnoty 0,5 (včetně) nahoru. V takovém případě je třeba při testech hraničních hodnot ověřit, že systém tuto skutečnost reflektuje a propouští/blokuje zadávané nabídky správně. A také posléze do tabulek i na obchodovací obrazovku zavádí skutečnou cenu nabídky zadanou uživatelem, nikoliv zaokrouhlenou, kterou vytváří pro porovnání s uživatelským limitem.

Pro lepší představu si opět rozepišme scénář na jednotlivé kroky, tentokrát zvolíme 5.2. PA pro testování horní meze uživatelských limitů (viz Obrázek 18 a Obrázek 19).

Zde se již jedná o scénář rozsáhlejší a náročnější, nejen co do počtu kroků, ale také rozsahu znalostí a prováděných kontrol. Kromě práce s testovanou aplikací je třeba, aby tester ovládal základy práce s databází a byl schopen na základě uvedených informací potřebná data ověřit. Předpokládaná délka tvorby i provádění tohoto typu scénáře se v ideálním případě pohybuje v desítkách minut.

5.2 PA Zavedení nabídky - horní hraniční hodnota uživatelských limitů

Popis:
- všechny údaje řádně vyplněny
- číselné hodnoty jsou nastaveny jako horní hraniční hodnoty uživatelsky stanovených limitů
- nabídka je odeslána do systému bez zobrazení informativní hlášky

	Testovací krok	Testovací data	Očekávaný výsledek	Vykonáno	Komentář testera
1.	Přihlaste se do aplikace jako běžný uživatel.	např. Uživatel 01	Uživatel úspěšně přihlášen	PASS	*Přihlášen Uživatel 01*
2.	Nastavte limit minimální ceny a množství pro komoditu "Brambory".	*Nastavení -> Uživatelské limity* Komodita: Brambory Max.cena: 20 Max. množství: 50 tlačítko *Uložit*	Hodnoty maximálního množství a ceny pro komoditu "Brambory" nastaveny.	PASS	*Limit nastaven podle popisu*
3.	Spusťte Obchodovací obrazovku.	*Obchodovací obrazovka*	Obchodovací obrazovka spuštěna.	PASS	
4.	Zaveďte do systému nabídku na nákup brambor.	*Komodita* : Brambory *Nákup* *Čas. int.* : 11-18 *Cena* : 20,2 *Množství* : 15 Odeslat	V panelu zavedení nabídky jsou vyplněny předepsané hodnoty. Nabídka je odeslána do systému.	PASS	*Panel zavedení nabídky vyplněn podle popisu. Nabídka odeslána.*
5.	Na Obchodovací obrazovce zkontrolujte zavedenou nabídku a zobrazené hodnoty		Nabídka je zavedena na obchodovací obrazovce. Data vyplněná v jednotlivých sloupcích odpovídají zadání a jsou uvedena ve správném tvaru	PASS	*Nabídka na obch. obrazovce zobrazena se stejnými hodnotami, jaké byly zadány v kroku 4.*
6.	Přihlaste se do databáze a zkontrolujte, že pro cenu i množství byly zapsány správné hodnoty.	DB *Jméno:* Uživatel 01 *Heslo:* Uživatel 01 *Tabulka:* Nabídky *select * from Nabídky t where t.Uzivatel = 'Uzivatel 01' and t.Interval = '11-18' and t.Komodita = 'Brambory'*	Nabídka byla v tabulce nalezena. Sloupce jsou naplněny správnými hodnotami včetně desetinných míst.	PASS	*Nabídka v DB nalezena dle uvedených parametrů. Hodnoty odpovídají kroku 4.*
7.	Zaveďte do systému nabídku na prodej brambor.	*Komodita* : Brambory *Prodej* *Čas. int.* : 8-13 *Cena* : 20,5 *Množství* : 15 Odeslat	V panelu zavedení nabídky jsou vyplněny předepsané hodnoty. Při odeslání nabídky do systému stiskem tlačítka je zobrazeno hlášení: *"Překročen limit maximální ceny, skutečně chcete nabídku odeslat?"*	PASS	*Panel zavedení nabídky vyplněn podle popisu.*
8.	V zobrazeném dotazu stiskněte "*Ano*" a ověřte, že se nabídka zavedla do systému.	tlačítko A*NO*	Nabídka byla zavedena do systému a ve stanoveném intervalu se zobrazuje na obchodovací obrazovce.	PASS	*Nabídka odeslána.*
9.	Na Obchodovací obrazovce zkontrolujte zavedenou nabídku a zobrazené hodnoty		Nabídka je zavedena na obchodovací obrazovce. Data vyplněná v jednotlivých sloupcích odpovídají zadání a jsou uvedena ve správném tvaru	PASS	*Nabídka na obch. obrazovce zobrazena se stejnými hodnotami, jaké byly zadány v kroku 7.*

Obrázek 18 - Testovací scénář Grey box (1. část)

10. Přihlaste se do databáze a zkontrolujte, že pro cenu i množství byly zapsány správné hodnoty.	DB *Jméno:* Uživatel 01 *Heslo:* Uživatel 01 *Tabulka:* Nabídky *select * from Nabídky t where t.Uzivatel = 'Uzivatel 01' and t.Interval = '8-13' and t.Komodita = 'Brambory'*	Nabídka byla v tabulce nalezena. Sloupce jsou naplněny správnými hodnotami včetně desetinných míst.	PASS	*Nabídka zapsána do DB s hodnotami odpovídajícími kroku 7.*
11. Zaveďte do systému nabídku na prodej brambor.	*Komodita* : Brambory *Prodej* *Čas. int.* : 9-11 *Cena* : 10 *Množství* : 50,5 Odeslat	V panelu zavedení nabídky jsou vyplněny předepsané hodnoty. Při odeslání nabídky do systému stiskem tlačítka je zobrazeno hlášení: *"Překročen limit maximálního množství, skutečně chcete nabídku odeslat?"*	FAIL	*Nabídka byla zavedena i přes to, že byl překročen nastavený uživatelský limit*
12. V zobrazeném dotazu stiskněte "*Ano*" a ověřte, že se nabídka zavedla do systému.	tlačítko A*NO*	Nabídka byla zavedena do systému a ve stanoveném intervalu se zobrazuje na obchodvací obrazovce.	BLOCK	
13. Na Obchodovací obrazovce zkontrolujte zavedenou nabídku a zobrazené hodnoty		Nabídka je zavedena na obchodovací obrazovce. Data vyplněná v jednotlivých sloupcích odpovídají zadání a jsou uvedena ve správném tvaru	BLOCK	
14. Přihlaste se do databáze a zkontrolujte, že pro cenu i množství byly zapsány správné hodnoty.	DB *Jméno:* Uživatel 01 *Heslo:* Uživatel 01 *Tabulka:* Nabídky *select * from Nabídky t where t.Uzivatel = 'Uzivatel 01' and t.Interval = '9-11' and t.Komodita = 'Brambory'*	Nabídka byla v tabulce nalezena. Sloupce jsou naplněny správnými hodnotami včetně desetinných míst.	BLOCK	
15. Zaveďte do systému nabídku na prodej brambor.	*Komodita* : Brambory *Prodej* *Čas. int.* : 12-13 *Cena* : 11 *Množství* : 50,9 Odeslat	V panelu zavedení nabídky jsou vyplněny předepsané hodnoty. Při odeslání nabídky do systému stiskem tlačítka je zobrazeno hlášení: *"Překročen limit maximálního množství, skutečně chcete nabídku odeslat?"*	PASS	
16. V zobrazeném dotazu stiskněte "*Ne*" a ověřte, že se nabídka NEzavedla do systému.	tlačítko NE	Nabídka NEbyla zavedena do systému a ve stanoveném intervalu se NEzobrazuje na obchodovací obrazovce.	PASS	*Nabídka se na obchodovací obrazovce nezobrazuje.*
17. Přihlaste se do databáze a zkontrolujte záznam nabídky z kroku 14	DB *Jméno:* Uživatel 01 *Heslo:* Uživatel 01 *Tabulka:* Nabídky *select * from Nabídky t where t.Uzivatel = 'Uzivatel 01' and t.Interval = '12-13' and t.Komodita = 'Brambory'*	Nabídka NEbyla v tabulce nalezena.	PASS	*Nabídka nebyla v DB nalezena.*

Obrázek 19 - Testovací scénář Grey box (2. část)

V této verzi scénáře můžeme vidět test, který proběhl s chybou. Byly však zablokovány pouze některé kroky. Jedná se tedy o situaci, kdy systém nereflektuje hraniční hodnotu množství, kde se láme zaokrouhlování, a nabídku propustí. Zřejmě tedy je třeba ověřit, zda pro horní hodnotu zadávaného množství je správně definován proces zaokrouhlování.

Ve scénáři si také můžeme povšimnout, že pokud bychom striktně následovali pouze předepsané kroky, ověříme celkem čtyři číselné hodnoty - dvě pro množství a dvě pro cenu. Ačkoliv jsou hodnoty voleny tak, aby testovaly hraniční hodnoty a tedy nejdůležitější rozhodovací momenty, nemohou pokrýt celý rozsah možných variant. Tento jev je přirozený a váže se k celé podstatě testování. Nikdy nelze ověřit vše. Proto není neobvyklým jevem, že se tester u těchto kroků pozastaví a neprovede je jednou, ale několikrát s různými čísly. Tyto opakované kroky a experimenty s hodnotami jsou většinou postaveny především na zkušenostech testera, který již ví, že tato konkrétní funkcionalita je na chyby náchylná a je třeba ji hlouběji prozkoumat nebo naopak.

Podobné experimentování je možné pochopitelně provádět i u testů Black box, v nich však bývá zadání jednoznačnější a není tedy nutné až tak do hloubky zkoumat celé chování.

5.4 White box test

Tento druh testů předpokládá nejhlubší znalost testovaného případu (viz 3.6 Testování metodou White box). V našem rozebíraném případě užití by tento druh testování odpovídal unit testům (jednotkovým testům), které provádějí programátoři po dokončení práce na určité funkcionalitě. Pro každou jednotlivou položku formuláře zavedení nabídky je třeba primárně ověřit, jestli reaguje tak, jak je očekáváno, a po té provést kontrolu, že všechny položky spolupracují a výsledkem je zavedená nabídka.

Sada jednotlivých unit testů je tvořena krátkými scénáři, tyto sady se obvykle užívají jako celek a jsou velmi vhodné k automatizaci. Zvláště u tak základních funkcí jako je popisovaný případ užití. Jedná se totiž o vstupní funkcionalitu systému, která je implementována jako jedna z prvních při vývoji a na její vstupy a výstupy jsou navazovány další úkony, které systém provádí. Při každém uvolnění nové verze je tedy třeba ověřit, že žádný původní proces nebyl narušen.

Sadu unit testů pro formulář zadání nabídky může vypadat například podle tabulky níže (viz Tabulka 3).

Tabulka 3 - Soubor unit testů pro panel Zadání nabídky

Test case	Test	Název
1	**Komodita**	
	1.1.	Ověření totožných položek v DB a v panelu Zadání nabídky, provedením změny položek v DB jsou změněny i položky v panelu
	1.2.	Provedení změny v DB vyvolá změnu v roletce panelu Zadání nabídky
	1.3.	Po klepnutí na položku v roletce je hodnota vybrána a vypsána jako zvolená
	1.4.	Pokud je položka ponechána prázdná, je zobrazena chybová hláška
2	**Nákup/Prodej**	
	2.1.	Vždy je možné zatrhnout právě jedno pole

Test case	Test	Název
3	**Časový interval**	
	3.1.	Hodnoty v obou roletkách obsahují 24 hodin
	3.2.	Po klepnutí na položku v roletce je hodnota vybrána a vypsána jako zvolená
	3.3.	Při nevhodné kombinaci intervalu je zobrazena chybová hláška
	3.4.	Je-li jedna nebo obě položky prázdné, je zobrazena chybová hláška
4	**Cena**	
	4.1.	Hodnotu lze vyplnit pouze na jedno desetinné místo
	4.2.	Je-li hodnota ponechána prázdná, je zobrazena chybová hláška
	4.3.	Hodnoty lze zadat pouze v stanoveném systémovém rozsahu 0-999 999
5	**Množství**	
	5.1.	Hodnotu lze vyplnit pouze na jedno desetinné místo
	5.2.	Je-li hodnota ponechána prázdná, je zobrazena chybová hláška
	5.3.	Hodnoty lze zadat pouze v stanoveném systémovém rozsahu 0-999
6	6.1.	Jsou-li vyplněny všechny položky řádně, je zobrazeno tlačítko odeslat
	6.2.	Je-li stanoven limit ceny, je provedena kontrola a při překročení zobrazen dotaz na odeslání
	6.2.	Je-li stanoven limit množství, je provedena kontrola a při překročení zobrazen dotaz na odeslání

Z uvedeného přehledu vidíme, že se jedná skutečně o základní testy, které by při rozepsání byly tvořeny jen několika málo kroky. Zde se však nejedná ani tak o ověření fungování na výstupní obrazovce, tedy v panelu zadání nabídky. Jedná se především o kontrolu kódu, propisování pokynů do databáze, čerpání dat z databáze a vazeb mezi jednotlivými položkami na formuláři. Uvedené kontroly vyžadují znalost kódu, znalost provedené analýzy a designu a tyto skutečnosti obvykle mezi sebou sdílí především tým analytiků a programátorů.

5.5 Automatizace testů

Všechny předchozí uvedené procesní testy pro daný případ užití jsou vhodné pro automatizaci. U jednotkových testů jsme již tuto skutečnost zmínili, u ostatních testů by se taktéž jednalo o rutinní kontrolu funkčnosti po nasazení nové verze software. Automatizace nám také umožňuje rychle a efektivně provést testy nejen pro namátkově zvolená data, ale pro celou škálu dat. Pokud je naprogramování jednotlivých testovacích scénářů vhodně provedeno, můžeme ve zlomku času (proti manuálnímu provádění) otestovat například všechny horní i dolní hraniční hodnoty uživatelských limitů pro množství i cenu, pro nákup i prodej a pro všechny dostupné komodity. Pokud bychom chtěli být až extrémní, mohli bychom experimentovat i s pokrytím všech (většiny) možných kombinací časového intervalu, ale tato variace se jeví jako irelevantní, neboť ovlivňuje pouze dobu viditelnosti nabídky na obchodovací obrazovce.

Uveďme si jako příklad následující situaci (viz Tabulka 4): vytvoříme tabulku ceny a množství nákupu a prodeje pro každou komoditu, která je v systému zavedena v libovolných časových blocích. Po té ji můžeme rozšířit o sloupce s limitem ceny

a limitem množství. Nakonec v tabulce vyznačíme ty položky, které jsou hraniční nebo neodpovídají stanoveným limitům. V tuto chvíli již na první pohled vidíme, že se dostáváme do velmi rozsáhlého množství testů k naplánování a provedení.

Tabulka 4 - Příklad hodnot pro automatické zakládání nabídek

Komodita	Interval	Prodej		Nákup		Min. cena	Max. cena	Min. množ.	Max. množ
		Cena	Množství	Cena	Množství				
Brambory	8-12	11,2	2,0	12,0	5,3	5	20	5	50
Cibule	13-18	18,9	13,5	18,6	12,4	8	15	4	20
Kukuřice	9-10	6,4	85,6	6,0	16,9	12	25	1	5
Mrkev	19-22	23,8	8,0	25,0	8,2	6	10	3	18
Okurky	8-15	14,3	6,1	10,9	11,0	10	30	2	12
Petržel	7-9	18,8	11,2	4,4	62,5	4	9	2	5
Řepa	10-13	8,0	58,6	18,0	14,0	12	24	5	10
Zelí	11-19	32,3	12,9	23,4	23,0	18	35	10	50

Základní testy bez uživatelského limitu:

- 8 testů pro jednotlivé komodity na prodej,
- 8 testů pro jednotlivé komodity na nákup.

Rozšířené testy s uživatelskými limity:

- 8 testů pro jednotlivé komodity pro prodej s horní mezí ceny a množství,
- 8 testů pro jednotlivé komodity pro prodej s dolní mezí ceny a množství,
- 8 testů pro jednotlivé komodity pro nákup s horní mezí ceny a množství,
- 8 testů pro jednotlivé komodity pro nákup s dolní mezí ceny a množství.

V tuto chvíli dostáváme 48 testů k provedení. Budeme-li navíc ještě uvažovat, že například pro komoditu „Brambory“ je pro nákup položka množství hraniční hodnota a tedy bychom chtěli provést samostatné testy pro hodnoty od 5,1 do 5,9, abychom viděli, že se pro jednotlivé desetiny provádí správně zaokrouhlení a vyhodnocení, rozšíříme počet testů o dalších osm na celkovou hodnotu 56.

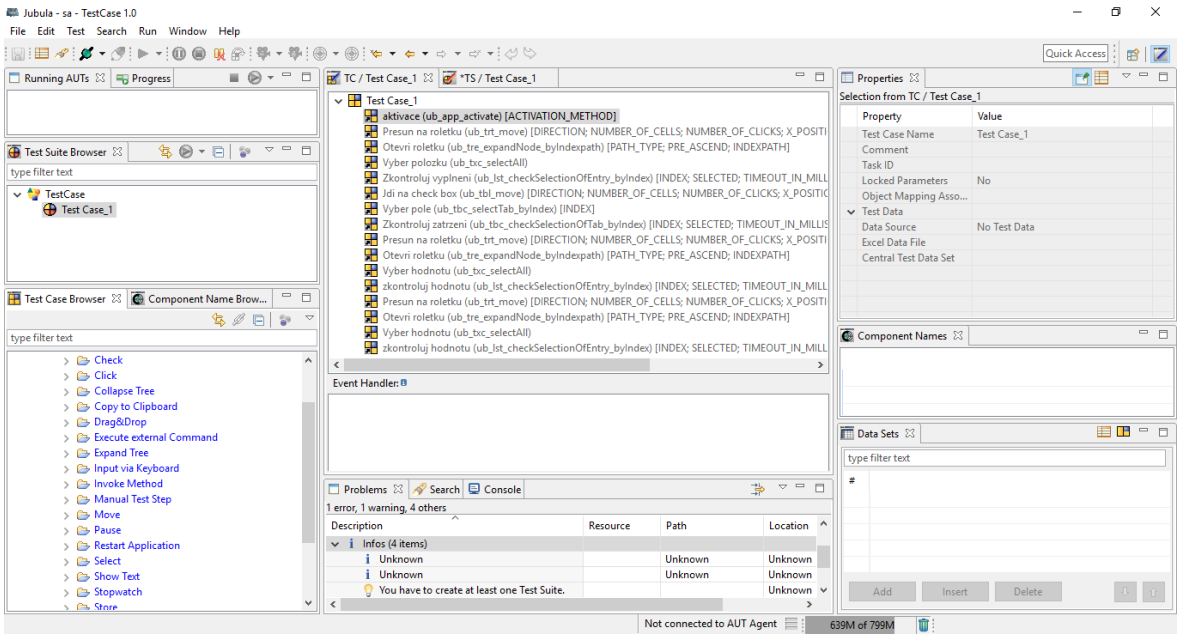
Vezmeme-li v úvahu skutečnost, že bychom stejné ověření chtěli provést také pro položky ceny nákupu a množství a ceny prodeje, alespoň pro jednu z komodit, vzroste počet testů o dalších 27 testů na celkový počet 83 testů.

Nyní, pokud budeme uvažovat manuální testování a zkušeného testera, který bude pracovat bez přestávky, a budeme předpokládat, že jeden test trvá přibližně 5 minut, dostáváme se na celkový výsledný čas 415 minut, tedy 6 hodin a 55 minut.

Naproti tomu jeden dobře naprogramovaný automatizovaný test může být zpracován v čase přibližně 5 sekund, čímž získáváme celkovým čas zhruba 7 minut.

Předvedli jsme si tedy časovou úsporu automatizovaných testů, ale abychom mohli z této výhody těžit, je třeba nejprve automatizované scénáře vytvořit a umět je formulovat tak, aby postihovaly veškeré naše požadavky a prováděly testování správně.

K ukázce automatického programování využijeme software Jubula (viz Obrázek 20). Pomocí identifikace jednotlivých grafických prvků testovaného formuláře a složení příkazů vztahujícím se k těmto grafickým prvkům do vhodného pořadí můžeme nasimulovat průběh testu stejným způsobem, jak je popsán v dříve zmíněné ukázce z Jira (Obrázek 17 - Testovací scénář Black box).



Obrázek 20 - Ukázka přípravy automatického testu v software Jubula

Jak můžeme již z prvního pohledu zaznamenat, není podoba automatizovaného testu ani zdaleka tak jednoduchá. Příprava spočívá v mnoha velmi malých krocích, které je třeba pomocí vhodně zvolených příkazů popsat a seřadit tak, aby přesně simulovaly chování, které v rámci testu chceme prověřit.

V levém dolním okně vidíme část knihovny příkazů sdružených podle povahy do složek. Z těchto příkazů na základě popisovaného kroku vybíráme a vytváříme v okně nahoře uprostřed jednotlivé úkony. Každý úkon je možno lépe specifikovat v pravém horním okně, případně doplnit o testovaná data v pravém dolním okně.

Na základě specifikovaných vstupních položek dokončíme přípravu testu navázáním vstupních a výstupních proměnných na reálná tlačítka, textová pole a výběrová menu v testovaném panelu.

Kromě vytváření celých testů je možné si také předpřipravit jen určité úkony, které víme, že budeme jen s drobnými modifikacemi využívat ve více různých testovacích případech, či jednotlivých testech. Tyto pak můžeme libovolně a v libovolném počtu zařazovat kamkoliv potřebujeme.

Abychom mohli automatizované testy evidovat stejně jako ostatní testy prováděné manuálně, použijeme nástroj Jenkins (viz Obrázek 21).

	Testovací krok	Testovací data	Očekávaný výsledek	Vykonáno	Komentář testera
1.	Přihlaste se do aplikace jako běžný uživatel.	např. Uživatel 01	Uživatel úspěšně přihlášen	PASS	*readLimitValues>openProductsMaintenance*
2.	Nastavte limit minimální ceny a množství pro komoditu "Brambory".	*Nastavení -> Uživatelské limity* Komodita Brambory Max.cena: 20 Max. množství: 50 tlačítko *Uložit*	Hodnoty maximálního množství a ceny pro komoditu "Brambory" nastaveno.	PASS	*readLimitValues>selectBrambory* *writeLimitValues>readMaxCena* *writeLimitValues>readMaxMnozstvi*

Obrázek 21 – Ukázka scénáře prováděného automatem

Jenkins je schopen navázat prováděné automatické scénáře, které spouští v software Jubula se scénáři, které jsou napsány v Jira pro běžné manuální testy. V reportech pak vidíme tyto scénáře, jako by byly provedeny a vyhodnoceny manuálně. Rozdíl, zda scénář prováděl tester či automat, je tak viditelný pouze v případě, že si otevřeme detail prováděného testu a prohlédneme si komentáře, kam si Jenkins značí pomocné informace.

Na základě výše uvedených informací můžeme tedy shrnout, že vytváření automatizovaných testů vyžaduje nejen testerské zkušenosti, ale také programátorský pohled a uvažování. Ačkoliv samotné vykonání testu, je-li dobře připraven, probíhá v řádu sekund, doba přípravy může (především v začátcích) vyžadovat až několik hodin a také opakované ladění a úpravy než dojde k překonání všech drobných nesrovnalostí a neočekávaných okolností narušujících hladký průběh testu.

5.6 Zhodnocení

Abychom mohli provést základní vyhodnocení manuálního a automatizovaného provedení testu z kapitoly 5.2 Black box test, shrneme si získané poznatky do tabulky (viz Tabulka 5). Uvedené hodnoty jsou pro zpracovávaný příklad užití průměrné a orientační (získané kvalifikovaným odhadem na základě dlouhodobého pozorování).

Tabulka 5 - Zhodnocení manuálního a automatizovaného Black box testu

	Manuální test	**Automatizovaný test**
Celkový čas přípravy testu	**20 minut**	**2 hodiny**
Celkový čas provedení testu	**5 a více minut** (podle vzniklých chyb)	**5 sekund**
Doba nutná pro dokumentaci nalezených chyb	**5 a více minut** podle komplikovanosti chyby a nutného dohledávání podkladů (aplikační logy, provázanost nalezené chyby na další akce, nová simulace chyby pro vizuální dokumentaci, popis nasimulování chyby, ...)	**0 sekund** neboť veškerá dokumentace probíhá jako součást prováděného testu a to s přesností záznamu na sekundy
Nezbytné znalosti a schopnosti testera (jeho „cena“)	**Tester junior**	**Tester senior** s přesahem do programování / **programátor**
Časové možnosti provádění testů	**Během pracovní doby testera**	**Kdykoliv** testy je možno spouštět nezávisle na pracovní době a přítomnosti testera/programátora

6 Závěr

Přípravu manuálních testovacích scénářů a jejich provádění lze považovat za poměrně intuitivní záležitost. Pochopitelně vše závisí na volbě využívaného nástroje a především na pochopení testovaného projektu a jeho funkční specifikace. Nicméně pokud si zvykneme na jazyk analytiků a obeznámíme se s testovaným softwarem, je práce na scénářích velmi prostá. Naším úkolem je stručně, jasně a srozumitelně popsat jednotlivé kroky, které vedou k očekávanému výsledku. Následné manuální testování představuje stejný proces, jen v opačném pořadí. Musíme být podle popsaných kroků schopni zopakovat to, co autor scénáře slovně popsal.

Samozřejmě nic není tak prosté a mnohdy může příprava i provedení testera pěkně potrápit, protože se jedná o složité kontroly, velké datové bloky nebo velmi specifické situace vyžadující precizní provedení. Přesto ale ve srovnání s automatizovaným testováním se stále jedná o postupy, které lze relativně jednoduše vysvětlit, předat, naučit.

Zabýváme-li se přípravou automatizovaných testů, zjistíme, že kromě pochopení testovaného software, funkční specifikace a programu, ve kterém chceme testy automatizovat, je zde ještě jedna zásadní myšlenka. Umět převést test, který hodláme provádět, do „jazyka“, kterému bude testovací program rozumět, tedy test naprogramovat.

Na první pohled to může vypadat, že pokud jsme pochopili princip, kterým pracuje nástroj pro automatizované testy, tak přece také víme, jakým jazykem s ním komunikovat při vytváření testů. Ale realita není tak jednoznačná. Chceme-li aby naše testy probíhaly řádně a obdržené výsledky byly plně relevantní, nesmíme opominout sebemenší maličkost. Automat totiž na rozdíl od testera nedomýšlí, dělá pouze to, co mu říká kód. Často pak testy mohou končit chybami nikoliv na základě skutečných problémů, ale kvůli velmi drobným nepřesnostem, které jsme pominuli při jejich přípravě.

Pokud například provádíme test, jehož součástí je spuštění nějaké obrazovky, musíme zapracovat čekací dobu odpovídající technologickému procesu načtení obrazovky. Tester obrazovku sleduje a ví, kdy je plně načtená a může přejít k dalšímu kroku. Automat potřebuje časovou známku a záchytný bod. Můžeme tedy například zavést čekání 3 vteřiny a kontrolu zobrazení nejpomaleji načítaného panelu dané obrazovky. Jakmile vyhodnotí tyto dva záchytné body kladně, může v testu pokračovat. Pokud v určitém nastaveném čase nedojde ke splnění, test se rovnou ukončí chybou.

Další problematickou částí automatického testování jsou neočekávané situace, které si při přípravě testu neuvědomíme. Máme například zadáno, že na začátku testu automat inicializuje testované okno kliknutím na jeho záhlaví a po té dole v okně vyplní kolonku nebo klikne na tlačítko. Test se ovšem nedaří provést a stále se vrací s chybným výsledkem. Příčinou je skutečnost, že při přesunu ukazatele, přejíždí tento přes lištu s roletkovými menu, která se automaticky vysouvají. A protože menu je natolik dlouhé, že překryje cílovou pozici, dochází ke kliknutí do menu místo na kolonku/tlačítko. Jelikož bylo toto chování roletek při přípravě opomenuto, test není proveden správně.

Stejný vliv mohou mít také neočekávaná hlášení (vyskakovací okna), které zablokují celou testovanou obrazovku a automat není schopen na ně řádně zareagovat.

A v neposlední řadě je třeba také zmínit, že automat není schopen kontrolovat vizuální podobu aplikace. V předchozí kapitole jsme zmiňovali skutečnost, že při testování nové části se nejprve ověří, zda vypadá tak, jak je očekáváno, a v dalších kolech testů je to pak pouze tester a jeho znalost, kdo může zaznamenat vzniklou odchylku.

Tuto problematiku automatizované testy nemohou pokrýt. Jsou zaměřeny na funkčnost a správné chování systému po stránce procesní.

V tuto chvíli to na základě výše uvedených fakt pravděpodobně vypadá, že dosud vychvalovaná automatizace, která nám má přinést mnoho výhod a pozitiv, je jen předražený problém (nutné vstupní investice, dlouhodobá návratnost), jež naši práci spíše zkomplikuje, než vylepší. Ale skutečnost je taková, že vše výše zmíněné lze považovat za jakési dětské nemoci, kterými je třeba projít. Po jejich překonání se dostaví ona slibovaná fáze urychlení a zefektivnění, na kterou budou především klást důraz všichni ti, kteří vás budou chtít k automatizaci nalákat.

Co si ale můžeme přesněji představit pod pojmem urychlení a zefektivnění. Úspora času je nepochybná, pomineme-li technologické prostoje způsobené během testovaného software, odehrávají se testy v rámci sekund. Efektivnost je pak přinášena na dvou frontách. Kromě skutečnosti, že mohou testy běžet nezávisle, bez dozoru a v kteroukoliv denní i noční dobu, jsou také dokonale dokumentovány.

V okamžiku, kdy se ve správně nastaveném testovacím scénáři vyskytne neočekávaná situace, má programátor k dispozici naprosto přesný záznam toho, co se v daný okamžik dělo. Může po sekundách sledovat probíhající proces, prohlédnout si obrazový záznam i záznam v logu. Této preciznosti nemůže nikdy manuální test docílit. Nejen, že tester není schopen reagovat tak rychle a v několika „frontách“ najednou, ale automatizovaný test může zachytit i chyby, které člověk ani nezaznamená, nebo je přikládá nějaké bezvýznamné shodě okolností, protože se jedná o takový mžik, že si ani není jist, jestli „něco viděl“. Automat ale i tuto chybu zaznamená, popíše a programátor ji může na základě těchto podkladů vysledovat a opravit, případně nechat automat provést novou simulaci, či nové simulace, aby zjistil, jestli se problém opakuje pravidelně, náhodně nebo šlo o velmi ojedinělý úkaz.

Závěrem tedy můžeme říct, že rozhodnutí o zapojení automatického testování do procesu vývoje vyžaduje především velmi důkladné zvážení a zodpovědné rozhodnutí. Pokud ale dokážeme uvolnit dostatek času a zdrojů na plné ovládnutí zvolených nástrojů, získané přínosy budou několikanásobně vyšší než původní vklad.

Poděkování

Děkuji Ing. Pavlu Smutnému, Ph.D. za vedení práce a veškerý věnovaný čas.

Ing. Janě Břenkové a kolegům nejen z testovacího týmu děkuji za umožnění studia, cenné rady a podporu.

Celé rodině a všem přátelům děkuji za podporu a především trpělivost a pochopení během celého studia.

7 Seznam použité literatury

ANON., , 2018a. *Metodika RUP a testování* [online]. [cit. 2018]. Dostupné z: http://test.swtestovani.cz/index.php?option=com_content&view=article&id=43:metodika-rup-a-testovani&catid=3:zaklady&Itemid=11

ANON., , 2018b. *Monkey business, Ad hoc testing, Free testing a zmatení pojmů* [online]. [cit. 2018]. Dostupné z: http://test.swtestovani.cz/index.php?option=com_content&view=article&id=26:monkey-bussiness-ad-hoc-testing-free-testing-a-zmateni-pojm&catid=13:testovaci-techniky&Itemid=29

ATLASSIAN, 2019. Atlassian. *Jira | Software pro sledování požadavků a projektů* [online]. [cit. 2019]. Dostupné z: https://cs.atlassian.com/software/jira

BOROVCOVÁ, Anna, 2008. *Testování webových aplikací*. Diplomová práce. Praha: Univerzita Karlova v Praze.

BUREŠ, Miroslav, Miroslav RENDA, Michal DOLEŽEL, Petr SVOBODA, Zdeněk GRÖSSL, Martin KOMÁREK, Ondřej MACEK a Radoslav MLYNÁŘ, 2016. *Efektivní testování softwaru*. Praha: Grada Publishing, a.s.

ČERMÁK, Miroslav, 2010a. Clever and smart. *Black box test* [online]. [cit. 2018]. Dostupné z: https://www.cleverandsmart.cz/black-box-test/

ČERMÁK, Miroslav, 2010b. Clever and smart. *White box test* [online]. [cit. 2018]. Dostupné z: https://www.cleverandsmart.cz/white-box-test/

ČERMÁK, Miroslav, 2010c. Clever and smart. *Gray box test* [online]. [cit. 2018]. Dostupné z: https://www.cleverandsmart.cz/grey-box-test/

ECLIPSE FOUNDATION, INC., 2019. Eclipse. *Eclipse Jubula Project | The Eclipse Foundation* [online]. [cit. 2019]. Dostupné z: https://www.eclipse.org/jubula/

HIGHSMITH, Jim, 2001. *Manifest Agilního vývoje software* [online]. [cit. 2018]. Dostupné z: http://agilemanifesto.org/iso/cs/manifesto.html

JULÍNEK, Pavel, 2008. *Použití RUP pro malé SW projekty*. Diplomová práce. Brno: Masarykova univerzita.

KAWAGUCHI, Kohsuke, 2019. Jenkins. *Jenkins* [online]. [cit. 2019]. Dostupné z: https://jenkins.io/

KNESL, Jiří, 2009a. Agilní vývoj: Úvod. *Zdroják, o tvorbě webových stránek a aplikací* [online]. [cit. 2018]. Dostupné z: https://www.zdrojak.cz/clanky/agilni-vyvoj-uvod/

KNESL, Jiří, 2009b. Agilní vývoj: Scrum. *Zdroják, o tvorbě webových stránek a aplikací* [online]. [cit. 2018]. Dostupné z: https://www.zdrojak.cz/clanky/agilni-vyvoj-scrum/

PATTON, Ron, 2002. Testování softwaru. PATTON, Ron. *Testování softwaru*. Praha: Computer Press.

RAMASUBBARAYALU, Shobika, 2017. Letzdotesting. *Software Testing Methods* [online]. [cit. 2018]. Dostupné z: http://letzdotesting.com/software-testing-methods/

ROUDENSKÝ, Petr, 2016. *Kvalita softwaru*. Prostějov: Computer Media s.r.o.

ROUDENSKÝ, Petr a Anna HAVLÍČKOVÁ, 2013. *Řízení kvality softwaru: průvodce testováním*. Brno: Computer Press.

STEINER, František, 2005. *Principy posuzování kvaliti software podle ČSN ISO/IEC 12119*. Praha. Prezentace. ZČU v Plzni, EZÚ Praha.

TECHNOR PRINT, s.r.o., 2018. ČSN ISO/IEC 9126-1 (369020) - Technické normy ČSN. *TECHNOR Ing. Jiří Řezníček* [online]. [cit. 2018]. Dostupné z: http://www.technicke-normy-csn.cz/369020-csn-iso-iec-9126-1_4_65526.html

USPENSKIY, Sergey, 2010. *A survey and classification of software testing tools* [Master's thesis]. Lappeenranta, s. 61 [cit. 2018]. Dostupné z: https://www.doria.fi/bitstream/handle/10024/63006/nbnfi-fe201005111842.pdf

VANÍČEK, Jiří, 2004. Cev.cemotel.cz. *Vlastimil Čevela - osobní profil a realizované projekty/aktivity* [online]. [cit. 2018]. Dostupné z: http://cev.cemotel.cz/programovani_a_tvorba_sw_1975-2004/2004/311.pdf

VONDRÁK, Ivo, 2002. *Úvod do softwarového inženýrství* [online]. Ostrava [cit. 2018]. Dostupné z: http://vondrak.cs.vsb.cz/download/Uvod_do_softwaroveho_inzenyrstvi.pdf

8 Příloha 1 - Slovník

Ad hoc testování	Testování prováděné neformálně; není k dispozici příprava formálních testů, nejsou použity žádné techniky návrhu testů, neexistují žádná očekávání ohledně výsledků a testy jsou prováděny dle libovolného uvážení.
Akceptační kritéria	Výstupní kritéria, která musí komponenta nebo systém splňovat proto, aby mohly být akceptovány uživatelem, zákazníkem nebo jinou oprávněnou osobou.
Agile manifesto	Agilní manifest; prohlášení o hodnotách, které jsou základem pro agilní vývoj software. Tyto hodnoty jsou: (preferování) jednotlivců a interakcí před procesy a nástroji, (preferování) reakce na změnu před sledováním plánu, (preferování) spolupráce se zákazníkem před vyjednáváním o smlouvě.
Agilní vývoj	Skupina metodik vývoje software založená na iterativním inkrementálním vývoji, kde se požadavky a řešení vyvíjejí během spolupráce v samo organizujících a mezi funkčních týmech.
Alfa testování	Simulované nebo skutečné testování prováděné v testovacím prostředí vývojářské organizace, ale lidmi mimo vývojový tým.
API	Application Programing Interface, aplikační programové rozhraní.
Automatizovaný test	Test prováděný strojem na základě předem vytvořeného postupu.
Beta testování	Simulované nebo skutečné produkční testování prováděné na externí straně lidmi mimo společnost, která produkt vyvinula.
Big bang testing	Metoda testování velký třesk; koncept integračního testování, ve kterém je integrace softwarových a/nebo hardwarových elementů do komponenty nebo celého systému upřednostňována před integraci v dané úrovni.
Black box testování	Funkcionální nebo nefunkcionální testování bez vztahu k vnitřní struktuře komponenty nebo systému.
Blocked	Blokovaný; označení stavu scénáře nebo jeho kroku v případě, že ne nelze z různých příčin provést.
Branch	Větev; základní blok (kódu), který může být vybrán pro provedení na základě konstrukce programu, ve které je k dispozici buďto jedna ze dvou anebo dvě nebo více alternativních cest programu, např. přepínač (case/switch), skok, příkaz goto nebo podmínka.
Buffer	Vyrovnávací paměť; zařízení nebo datové úložiště, které slouží k dočasnému ukládání dat z důvodu rozdílných rychlostí toku dat, času nebo výskytu událostí. Důvodem pro využívání vyrovnávací paměti může být i rozdíl v množství dat, které jsou schopny zpracovat zařízení nebo procesy podílející se na jejich přenosu či použití.
Bug	Chyba, incident; jakákoliv událost, která vyžaduje prozkoumání.
Burnedown chart	Graf burn-down; zveřejněný graf, který znázorňuje práci v závislosti na čase v dané iteraci. Zobrazuje stav a trend při plnění úkolů v iteraci. Osa X typicky reprezentuje dny v iteraci, osa Y reprezentuje zbývající práci (obvykle buď v ideálních člověko-hodinách nebo v tzv. story points).

Certifikace	Proces potvrzení (např. složením zkoušky, provedením testů), že komponenta, systém, software, hardware nebo osoba je ve shodě se specifikovanými požadavky.
Dashboard	Reprezentace dynamických měření provozní výkonnosti nějaké organizace nebo činnosti za použití metrik, které jsou metaforou prvků na palubní desce automobilu, jakými jsou například vizuální číselník, čítač, apod. Důsledek událostí nebo činností tak může být snadno pochopen v souvislosti s provozními cíli.
Datový tok	Abstraktní znázornění posloupnosti a možných změn stavu datových objektů, kde stav objektu je vytvořený, použitý nebo zrušený.
Debugger	Nástroj pro ladění umožňující programátorům spouštět programy krok za krokem, přerušit běh programu na libovolném příkaze, nastavit a přezkoumat proměnné programu.
End-to-end testy	Průchod sledovaného prvku po celou dobu jeho životnosti napříč systémem.
Exploratorní testování	Průzkumné testování; přístup k testování založený na zkušenostech, kdy tester dynamicky (teprve v průběhu samotného testování) navrhuje a provádí testy, a to na základě svých znalostí (často i intuice), průběžného stavu testované položky a výsledků předchozích testů.
Fail	Selhání; označení stavu scénáře nebo jeho kroku, test selhal, pokud jeho skutečný výsledek neodpovídá výsledku očekávanému.
FS	Funkční specifikace.
Funkcionalita	Schopnost softwarového produktu poskytovat funkce, které uspokojí stanovené a předpokládané potřeby, pokud je software používán za specifikovaných podmínek.
Grey box test	Test prováděný na základě omezené znalosti interních datových a programových struktur.
GUI	Grafické uživatelské rozhraní.
Integrační testování	Testování prováděné s cílem odhalit chyby na rozhraních a v interakcích mezi integrovanými komponentami nebo systémy.
Kód	Počítačové instrukce a definice dat vyjádřené v programovacím jazyce nebo ve formě výstupu z asembleru, kompilátoru nebo jiného překladače.
Kódovací standard	Formátování kódu jednotným způsobem.
Komponenta	Nejmenší část systému, kterou lze izolovaně testovat.
Kritická funkcionalita	Funkcionalita nezbytná pro chod programu, obvykle bývají implementovány ve vývoji jako první, tvoří kostru projektu. Na ně jsou postupně nabalovány další nekritické funkcionality.
LCSAJ	Zkratka pro sekvence lineárního kódu a skoků. Skládá se z následujících tří položek (dle zažité konvence se řádky zdrojového kódu číslují): začátek lineární posloupnosti vykonatelných příkazů; konec této lineární posloupnosti; a cílový řádek, na který je řídící tok na konci této lineární posloupnosti přenesen.
Manuální test	Test prováděný ručně určenou osobou krok po kroku.
Metrika	Stupnice měření a metoda definovaná pro její měření.

Migrace dat	Proces kompletního přenosu dat z jednoho systému do druhého.
Modulové testy	Nejnižší úroveň testování prováděná testovacím týmem. Jsou testovány základní stavební bloky budoucího programu.
Nekritická funkcionalita	Funkcionality, které nemají zásadní význam pro chod programu.
Quality assurance	Zajištění kvality; část řízení kvality zaměřená na poskytování důvěry, že požadavky na kvalitu budou naplněny.
Pass	Prošel; označení scénáře nebo jeho kroku, test prošel (úspěšně), pokud jeho skutečný výsledek odpovídá výsledku očekávanému.
Patch coverage	Pokrytí cesty; procento cest, které je vykonáno v rámci testovací sady, 100% pokrytí cest znamená 100% LCSAJ pokrytí.
Pokrytí kódu	Analytická metoda, která určuje, jaké části software byly prověřeny (pokryty) testovací sadou, a které části nebyly prověřeny, např. pokrytí příkazů, pokrytí rozhodnutí nebo pokrytí podmínek.
Regresní testování	Testování již dříve testované komponenty nebo systému po modifikaci s cílem zajištění toho, že nedošlo na nezměněných částech software v důsledku provedených změn k výskytu nových nebo dříve neodhalených (skrytých) defektů.
RUP	Rational Unified Process; vlastní přizpůsobivý iterativní rámec procesu vývoje software, který se skládá ze čtyř fází životního cyklu projektu: založení (inception), zpracování (elaboration), provedení (construction) a převedení (transition).
Scrum	Iterativně inkrementální rámec pro řízení projektů běžně používaný v agilním vývoji software.
Smoke test	Podmnožina všech definovaných/plánovaných testovacích případů, které pokrývají hlavní funkcionalitu komponenty nebo systému. Jejich úspěšné provedení potvrzuje funkcionalitu nejdůležitějších funkcí komponenty nebo systému, nejsou řešeny jemnější detaily.
State coverage	Pokrytí příkazů; testování je navrženo tak, aby došlo ke spuštění všech příkazů v programu alespoň jednou.
Testování software	Empirický technický výzkum kvality testovaného produktu nebo služby prováděný za účelem poskytnutí těchto informací všem zainteresovaným osobám. Testování je tedy zejména o hledání určitých informací o produktu jeho zkoumáním.
TC	Test case, testovací případ; sada vstupních podmínek, vstupů, očekávaných výsledků, výstupních podmínek a případně akcí, která je vypracována na základě testovacích podmínek.
Test execution	Provedení testů; proces, při němž dojde k otestování komponenty nebo systému, a jenž poskytne aktuální výsledek nebo výsledky.
UAT	User acceptance test, uživatelské akceptační testy; obvykle probíhají před předáním hotového produktu zákazníkovi. Slouží k ověření, že vzniklý produkt splňuje dohodnuté parametry.
UT	Unit test, jednotkové testy; obvykle prováděny automatizovaně přímo vývojáři. Testují se nejmenší programové části před tím, než jsou spojovány do modulů.

UC	Use case, případ užití; posloupnost operací s hmatatelným výsledkem, která je vyjádřena dialogem mezi aktérem a komponentou nebo systémem, kde aktér může být uživatel nebo cokoliv, co si může vyměňovat informace se systémem.
User story	Uživatelský scénář; Obecný uživatelský nebo bussines požadavek obvykle používaný v agilním vývoji software, který je typicky popsán jednou větou v každodenním nebo bussines jazyce. Snaží se podchytit, jakou funkcionalitu uživatel potřebuje, a to včetně důvodů k tomu vedoucích.
White box test	Testování metodou bílé skříňky založené na analýze vnitřní struktury komponenty nebo systému.
WIP	Work in progress, probíhající práce; označení stavu testovacího scénáře, nebo jeho kroku.
XP	Extreme programming; metodika softwarového inženýrství používaná v agilním vývoji software, ve které jsou hlavními praktikami programování v párech, provádění rozsáhlých revizí, jednotkové testování, jednoduchost a přehlednost veškerého kódu.
Zaslepené prostředí	Testovací prostředí, které není napojeno na všechny moduly, s nimiž běžně spolupracuje, nebo v budoucnu spolupracovat bude. Zaslepená část se během testů buď nevyužívá, nebo je nahrazena simulací budoucího procesu.
Životní cyklus software	Časový interval, který začíná, když je softwarový produkt formován a končí, když již není k dispozici k užívání. Životní cyklus software typicky zahrnuje fáze konceptu, požadavků, návrhu, implementace, testování, instalace a vyzkoušení, provozu a údržby a někdy také fázi stažení (z produkce). Tyto fáze se mohou překrývat nebo mohou být vykonávány iterativně.